VOL. 4 NO. 12 DEC. 1987

COMPUTER DIGEST

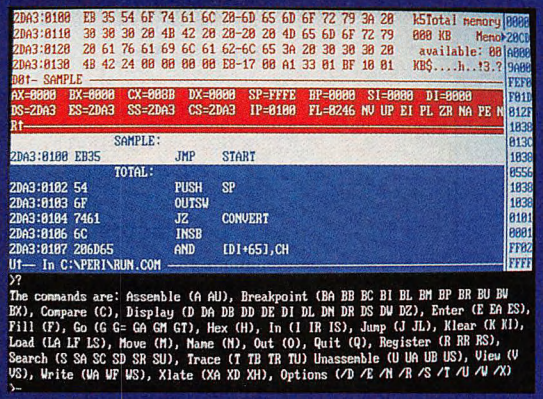
A NEW KIND OF MAGAZINE FOR ELECTRONICS PROFESSIONALS

FLOPPY DISKS

Bits, bytes, and copy protection

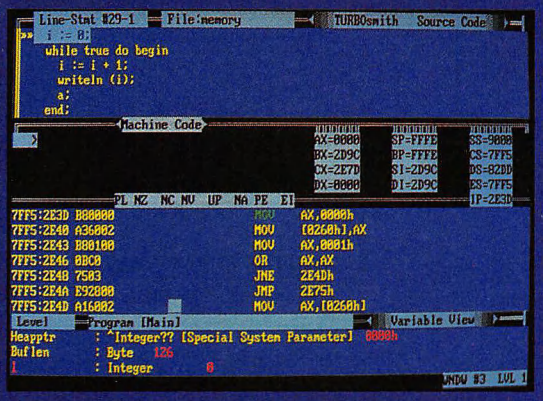
68000 COMPUTING

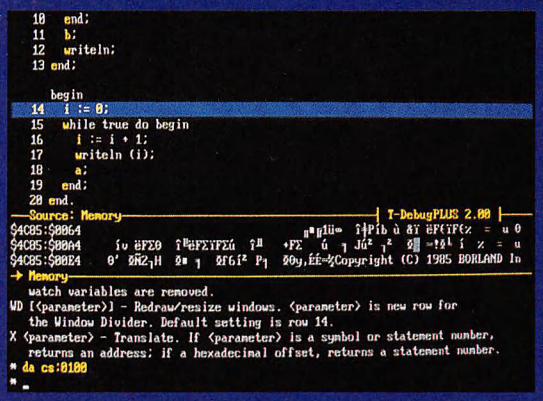
The big moment


2DA3:0100 EB 35 54 6F 74 61 6C 20-6D 65 6D 6F 72 79 3A 20 ▌5Total memory
2DA3:0110 30 30 30 20 4B 42 20 20-20 20 4D 65 6D 6F 72 79 000 KB Memory
2DA3:0120 20 61 76 61 69 6C 61 62-6C 65 3A 20 30 30 30 20 available: 000
2DA3:0130 4B 42 24 00 00 00 00 E8-17 00 A1 33 01 BF 10 01 KB$....h..!3.?
D0↑- SAMPLE
AX=0000 BX=0000 CX=008B DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=2DA3 ES=2DA3 SS=2DA3 CS=2DA3 IP=0100 FL=0246 NV UP EI PL ZR NA PE N
Rt
SAMPLE:
2DA3:0100 EB35 JMP START
TOTAL:
2DA3:0102 54 PUSH SP
2DA3:0103 6F OUTSW
2DA3:0104 7461 JZ CONVERT
2DA3:0106 6C INSB
2DA3:0107 286D65 AND [DI+65],CH
Ut— In C:\PERI\RUN.COM
>?
The commands are: Assemble (A AU), Breakpoint (BA BB BC BI BL BM BP BR BU BW BX), Compare (C), Display (D DA DB DD DE DI DL DN DR DS DW DZ), Enter (E EA ES), Fill (F), Go (G G= GA GM GT), Hex (H), In (I IR IS), Jump (J JL), Klear (K KI), Load (LA LF LS), Move (M), Name (N), Out (O), Quit (Q), Register (R RR RS), Search (S SA SC SD SR SU), Trace (T TB TR TU) Unassemble (U UA UB US), View (V VS), Write (WA WF WS), Xlate (XA XD XH), Options (/D /E /N /R /S /T /U /W /X)
>-

DEBUGGING TOOLS

Hands-on report

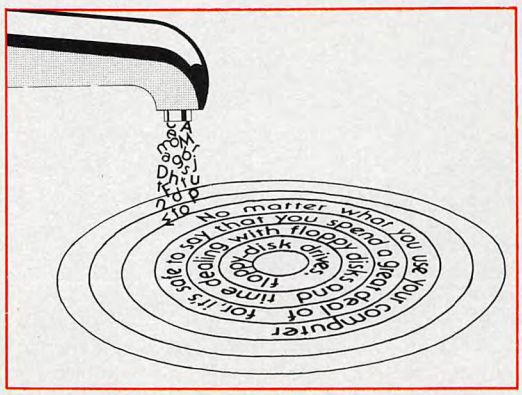



10 end;
11 b;
12 writeln;
13 end;
begin
14 i := 0;
15 while true do begin
16 i := i + 1;
17 writeln (i);
18 a;
19 end;
20 end.
Source: Memory T-DebugPLUS 2.00
$4C85:$0064
$4C85:$00A4
$4C85:$00E4
Memory
watch variables are removed.
WD [<parameter>] - Redraw/resize windows. <parameter> is new row for the Window Divider. Default setting is row 14.
X <parameter> - Translate. If <parameter> is a symbol or statement number, returns an address; if a hexadecimal offset, returns a statement number.
* da cs:0100
*

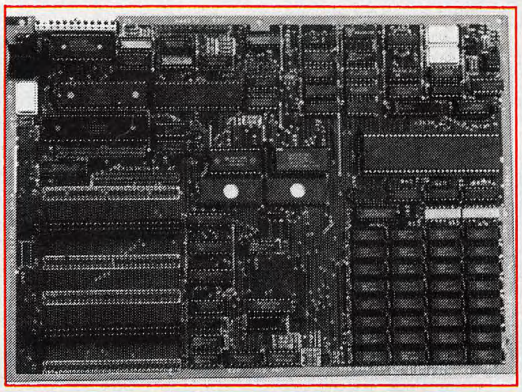
A GERNSBACK PUBLICATION

CONTENTS DECEMBER 1987

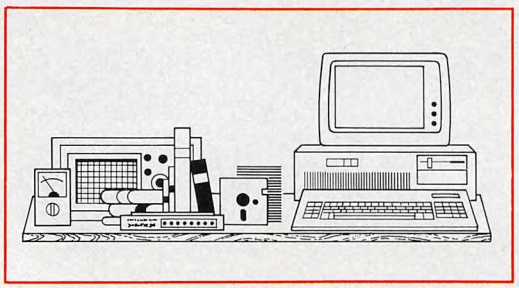
Vol. 4 No. 12


No matter what you use your computer for it's safe to say that you spend a great deal of time reading from and writing to floppy disks.
floppy disk data

91 ALL ABOUT FLOPPY DISKS
Focus on Apple and IBM formats.



95 CD CLASSROOM, PART 3
Install the microprocessor.



87 EDITOR'S WORKBENCH
HARDWARE: Periscope
SOFTWARE: TDebug Plus
TurboSmith

COMPUTER DIGEST

Larry Steckler,
EHF, CET: publisher & editor in chief

Art Kleiman,
editorial director
Brian C. Fenton,
managing editor
Jeff Holtzman
technical editor
Byron G. Wels,
associate editor
Carl Laron,
associate editor
Robert A. Young,
assistant editor
Teri Scaduto
editorial assistant
Ruby M. Yee,
production director
Karen Tucker,
production advertising
Robert A. W. Lowndes,
production associate
Marcella Amoroso
production assistant
Andre Duzant,
technical illustrator
Jacqueline P. Cheeseboro
circulation director
Arline R. Fishman,
advertising director

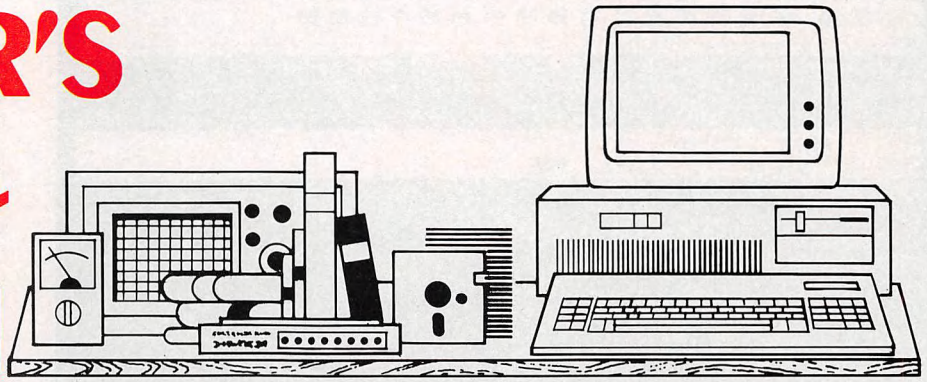
ComputerDigest
Gernsback Publications, Inc.
500-B Bi-County Blvd.
Farmingdale, NY 11735

ADVERTISING SALES 516-293-3000
Larry Steckler
Publisher

NATIONAL SALES
Joe Shere
1507 Bonnie Doone Terrace
Corona Del Mar, CA 92625
714-760-8697

Cover Photography by Herb Friedman
and Jeff Holtzman

EDITOR'S WORK-BENCH



Periscope

Debug is the standard MS DOS machine-language debugger. It is included with every copy of DOS, so every programmer, hacker, and hobbyist is familiar with it. Debug is useful for tracing and analyzing both software you write yourself, and software written by others. And with care you can even use it to create short programs without the aid of an external assembler.

However, you don't have to use the program for very long to realize that it has limitations, the most severe of which are limited breakpoint facilities, lack of windowing, and lack of support for the symbols generated by the assembler or compiler you use to generate object code.

Attempting to remedy those deficiencies, the Periscope Company sells a line of high-quality debugging tools with varying capabilities. The latest incarnation includes an expansion card that allows you to trap bus events in real time, thereby providing many of the capabilities of a development system for about 10% of the cost.

Actually, there are four models of Periscope, which differ mainly in the hardware included.

- Periscope I ($345) includes a memory board with 56K of write-protected memory and a break-out switch. The memory is installed outside of the 640K DOS limit (usually in segment D000 or E000); the Periscope software runs from that memory. Pressing the break-out switch at any time—even when the system is apparently locked-up—generates a hardware interrupt that activates the Periscope software.
- Periscope II ($175) includes just the break-out switch; the software runs from normal DOS memory, so it may be overwritten by a run-away program. In that case you may not be able to regain control of a hung system. The break-out switch does not require an expansion slot; a small finger slips in an in-use slot between the expansion card and the connector's own finger.
- Periscope II-X ($145) includes just the software and no hardware. You may use your own break-out switch or a keyboard hot-key to activate the software.
- Periscope III ($995) is a full-length expansion card (shown in Fig. 1) that includes 64K of write-protected memory, a break-out switch, and additional circuitry that continuously monitors the expansion bus and allows you to create breakpoints that are activated when memory locations, I/O ports, or microprocessor registers attain certain values. A small module must be inserted into your motherboard's 8087 (or 80287) socket.

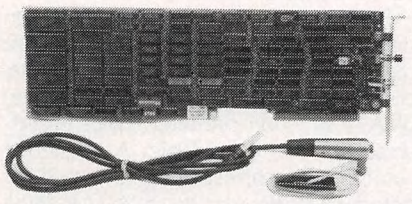


FIG. 1—The Periscope board.

The software functions pretty much the same for all versions, given the hardware differences. In fact, the same software is included with all versions; during the installation you must specify your hardware configuration.

How to use it

The Periscope software comes in two parts: PS.EXE, the 50K debugger, and RUN.COM, a 7K program that loads and runs your software. First you run PS, which lies dormant in memory until you Run a program. When you do, the Periscope screen comes up, a register dump is displayed, and the first instruction is disassembled. You can then execute your program in various ways.

To ease the learning process, Periscope duplicates many Debug commands, so you can Trace a single instruction or a group thereof, or Go, starting at an address you specify, with breakpoints at addresses you specify. In addition, you can Dump memory in the standard Debug format, Read and Write files and disk sectors, Unassemble, Examine, Search, and Fill memory, etc.

Normally, all output scrolls up the screen in standard fashion. However, you can open windows (see this month's cover for an example) that maintain information in fixed places on the screen, thereby making that information much easier to find and assimilate. Window size and color is easily specified; windows may contain registers, memory dumps (in several formats), disassembled instructions, stack dump, and a text file dump. Pressing Ctrl-F9 or Ctrl-F10 brings up default windows for mono and color windows, respectively.

The disassembly will contain the symbols and high-level instructions of your program, if an appropriate .MAP file is available. (Most assemblers and linkers can generate such a file.) For example, the program shown in Fig. 2 is written in Turbo Pascal; a utility sold by Turbo Power created the required .MAP file. Note that both Turbo source lines and the generated object code are shown. See this month's software reviews for more information.

You can add your own labels to programs and save them separately. That capability is useful when disassembling third-party software.

Breakpoints and program tracing

You can set an almost bewildering variety of breakpoints with Periscope. BC sets a Code Breakpoint, activated when your program tries to execute code at the specified address. That address may be specified using hex notation or symbolically. BI sets an Interrupt Breakpoint, activated when a software interrupt is executed. For example, to stop execution any time a DOS function (interrupt 21h) was to be executed, you'd type

```
BI 21
```

You can also set breakpoints on source-

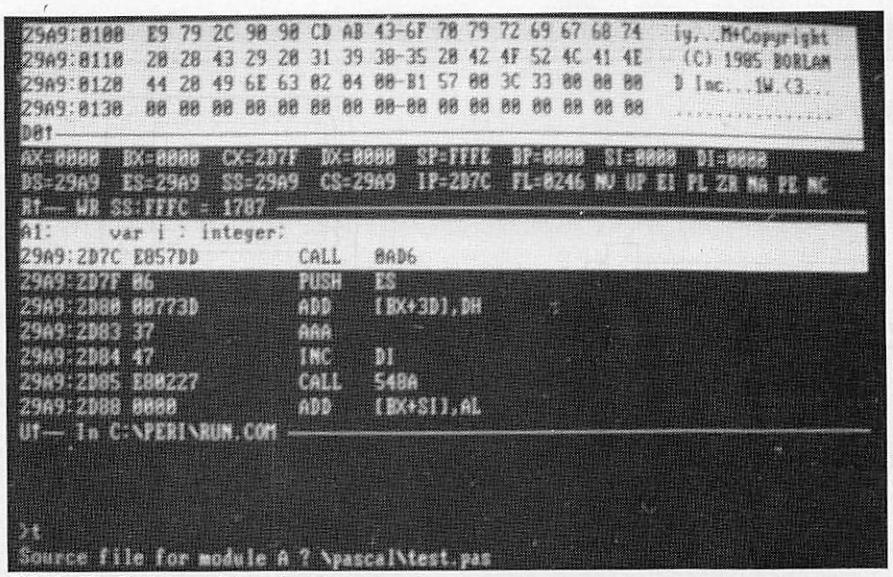


FIG. 2—The Periscope debugging screen.

code line numbers (BL), memory addresses (BM), I/O port addresses (BP), registers (BR), user-specified conditions (BU), memory contents (BW), and subroutine exit (BX). Note that those are not hardware-activated breakpoints, so execution must be started via the GT command, in which machine instructions are executed one at a time, breakpoint settings are examined, and then control reverts either to Periscope or to the user program, depending on whether the breakpoint conditions were met.

Each of the breakpoint commands may be cleared by entering the command with an asterisk. In addition, breakpoints of a particular type may be enabled (by typing +), disabled (by typing −), or displayed (by typing ?).

Hardware control

All of the commands discussed thus far may be used on any version of Periscope. Periscope III enables an additional set of breakpoint, trace, and display commands that let you execute your program in real time until a specified condition is met. The HB command, for example, sets a breakpoint when a specific bit pattern appears on the data bus. Values of 0, 1, and X (don't care) are legal. You can also trap memory and port reads or writes (or either) using the HM and HP commands, respectively. Further, using the JD command you can qualify the breakpoint so that execution will continue until data within a specified range, and at the specified address is accessed.

Periscope III's hardware buffer has sufficient memory to trap 8192 bus events. When defining a breakpoint, you can instruct the hardware to save the 8K events preceding the breakpoint, the 8K following it, or 4K on either side. You can also set up a pass count, so that, for example, the breakpoint will not be executed until the fifth time an 0Dh is output to port 61h. The HC (Hardware Control) command sets those modes.

Operation on an AT is somewhat more complicated. For example, when setting a bit breakpoint, you must specify the upper or lower half of the bus (or both). In addition, you can force a hardware breakpoint to be executed when memory beyond the one-megabyte area is accessed, again using the HC command.

Hardware buffer display

There are four commands for displaying the contents of the hardware buffer; a fifth (DW) saves the buffer to disk (or memory) for future examination. The display commands are as follows.

The HR command provides a full-screen display of the raw contents of the hardware buffer. Each line contains an address, data, operation type (input, output, read, write, DMA, instruction pre-fetch), plus an address symbol, if available. A sequence number that corresponds to the 8K of stored data is also displayed. You can scroll through the buffer and search for address, data, and operation types.

HS displays a single line in the same format as the HR command.

HT and HU provide a full-screen display of the hardware buffer, but in a more digestible format. HT provides a disassembly (with labels) plus the operation types (in, out, etc.), and HU provides just the disassembly.

Other goodies

Naturally, Periscope includes a built-in assembler. The surprise is that you can use symbols for address references—those from your program's MAP file, or those that you've defined within Periscope (using ES).

There is also (optional) on-line help. Pressing ? brings up a list of commands; pressing ? plus a command brings up a brief summary of that command.

Periscope can use two monitors—one for your program's output, and one for the Periscope debug display. When used with an EGA and enhanced color display, the Periscope display can optionally run in the 43-line mode.

The Periscope command line is fully editable, and the program maintains a stack of recently issued commands, which you can scroll through and then edit and execute the new line. Limited macro-type facility is available, and you can repeat the last command executed by pressing F4.

A word about the break-out switch: If you run the following piece of code under Debug, the only way to regain control of your machine would be by resetting:

XXXX:0100 JMP 0100

A break-out switch will, however, allow you to escape from the loop.

Conclusions

With or without a break-out switch, Periscope impresses us as one of the finest pieces of development software we've ever seen. It will be part of our tool kit for years to come. The hardware models aren't cheap, but if you need one, you need it bad, so cost should be no object. If you're on a tight budget, you can buy the software-only version and add your own break-out switch.

Turbo Pascal Debuggers

Turbo Pascal, when it was introduced some four years ago, laid to rest forever the notion that serious software development could only be done by means of the traditional and time-consuming edit-compile-link-test-repeat loop. Comprising less than 40K of code, and widely available at well under $100, Turbo Pascal also dispelled the notion that a professional compiler had to be big or expensive. In addition, it's available for a number of operating systems (CP/M, CP/M-86, and MS-DOS).

The only thing missing from Turbo Pascal was a symbolic source-level debugger of the type that companies like Periscope (see this month's hardware review) have been selling for some time.

That lack was addressed a few years ago when a program called TDebug appeared on BBSes across the nation. Since then, the product has gone commercial; the commercial implementation, called T-DebugPlus, adds a host of new features and several extremely useful utility programs.

TurboSmith, a latecomer, provides many of the features of T-DebugPlus, as well as several unique ones, including an integrated machine-language debugger.

TurboSmith

The program is contained in a single 120K file (TSM.EXE). In keeping with the Turbo philosophy, TurboSmith costs $69. It's writ-

ten entirely in assembly language, and requires 512K of free RAM to run. In fact, TurboSmith really needs most of 640K to function properly; it wouldn't work at all on a 3Com network station, which gobbles about 200K of memory for device drivers.

You run TSM to get into Turbo; TSM loads Turbo and adds a new item, Trace, to Turbo's main menu. (See Fig. 3.) The editor and compiler functions work just as they do without TSM; the new Trace option compiles the current program and then brings up the debugging screen. (See Fig. 4.)

That screen can display as many as three windows simultaneously. One is the source-code window, which is always displayed. The other two may contain program variables, a hex/ASCII memory dump, or a machine-language control window.

You switch among windows cyclically by pressing the F10 key; pressing F2 inside any window brings you to a command line where you can initiate various operations. For example, by typing Q you return to the Turbo menu. A full range of line-editing functions is available at each command line.

Other non-command-line operations are initiated by Ctrl-key and Alt-key combinations, and the commands work as much alike as possible inside different windows. For example, pressing Alt-B inside either the source or the machine-code window sets a breakpoint where the program will stop when it tries to execute that Pascal statement or that machine-language instruction, respectively.

In either program-control window you can execute a single statement or instruction by pressing the + key by the numeric keypad. Or press Alt-X (eXecute) to execute full-speed until a breakpoint is reached or Alt-S (Stop) is pressed. Another way of starting executing is to press Alt-A (Auto single-step). In that mode, a statement or instruction is executed, the screen is updated, and then the next program element is executed. Terminate Auto mode by pressing any key. Auto mode is not as useful as it might be because you can't control the speed at which statements are executed. You might want to slow execution in order to get the feel of how a loop was executing, for example.

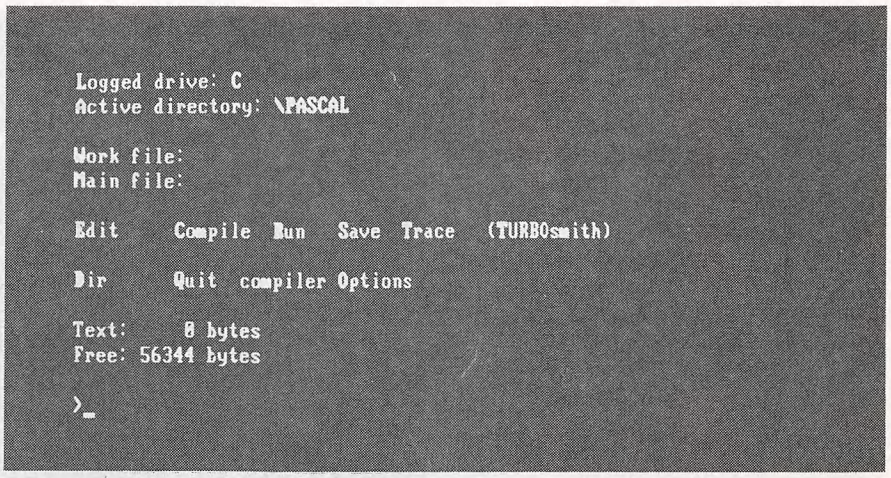


FIG. 3—TurboSmith adds a new item to the Turbo menu.

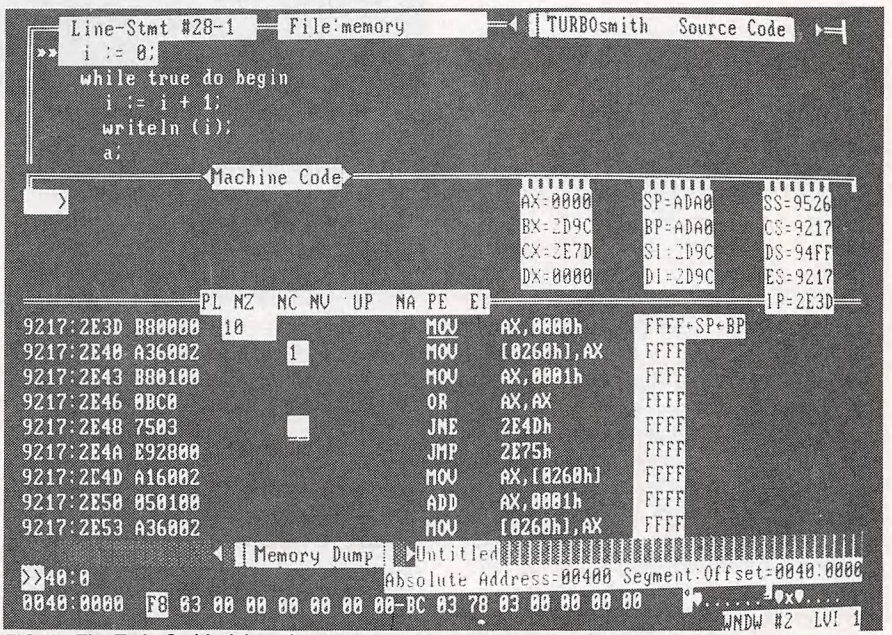

Line-Stmt #28-1
File:memory
TURBOsmith Source Code
i := 0;
while true do begin
i := i + 1;
writeln (i);
a;
Machine Code
AX=0000 SP=ADA0 SS=9526
BX=2D9C BP=ADA0 CS=9217
CX=2E7D SI=2D9C DS=94FF
DX=0000 DI=2D9C ES=9217
PL NZ NC NV UP NA PE EI IP=2E3D
9217:2E3D B80000 10 MOV AX,0000h FFFF:SP+BP
9217:2E40 A36002 1 MOV [0260h],AX FFFF
9217:2E43 B80100 MOV AX,0001h FFFF
9217:2E46 0BC0 OR AX,AX FFFF
9217:2E48 7503 JNE 2E4Dh FFFF
9217:2E4A E92800 JMP 2E75h FFFF
9217:2E4D A16002 MOV AX,[0260h] FFFF
9217:2E50 050100 ADD AX,0001h FFFF
9217:2E53 A36002 MOV [0260h],AX FFFF
Memory Dump
Untitled
>>40:0
Absolute Address=00400 Segment:Offset=0040:0000
0040:0000 F8 03 00 00 00 00 00 00-BC 03 78 03 00 00 00 00
WNDW #2 LV! 1

FIG. 4—The TurboSmith debugging screen.

The Variables window displays the values of all global variables, Turbo's internal variables as well as your own. You can only view variables inside a nested procedure or function when the program is executing that procedure or function. Further, you can open as many as eight window *levels*, each of which overlays one of the three on-screen windows. You cycle among displayed windows by pressing F10, and among levels by pressing F9. You are free to change the values of variables within any window.

Version 2.1 (which was not available in time for this review) should automatically open variable windows in nested procedures and functions.

Screen swapping

Most programs produce some sort of screen output; TSM can deal with screen output in two ways. First, and best, you can use two monitors, one mono (IBM or Hercules) and one color (CGA or EGA). In that case, program output must go to the mono screen; the debugging windows come up on the color screen. Otherwise, on a single-monitor system, you can use screen swapping, wherein TSM maintains memory images of both the debugging data and the program's output data, swapping them at the press of a key.

One useful feature is that TSM will respond to a break-out switch. If your program gets locked in a loop and the keyboard won't respond, chances are the break-out switch will allow you to regain control and continue debugging. See this month's hardware review for more information on break-out switches.

TSM's machine-language debugger provides powerful breakpoint facilities. You

Products Evaluated

- TurboSmith ($69), Visual Age, 642 N. Larchmont Blvd., Los Angeles, CA 90004, 800-732-2345, 213-534-4202 (CA).
  CIRCLE 31 ON FREE INFORMATION CARD
- T-DebugPlus ($60), Turbo Power, 2109 Scotts Valley Drive, Scotts Valley, CA 95066, 408-438-8608.
  CIRCLE 32 ON FREE INFORMATION CARD
- Turbo Pascal ($99.95), Borland International, 4585 Scotts Valley Drive, Scotts Valley, CA, 800-255-8008, 408-438-8400 (CA).
  CIRCLE 33 ON FREE INFORMATION CARD
- Periscope (price varies according to model), The Periscope Company, 14 Bonnie Lane, Atlanta, GA 30328, 404-256-3860.
  CIRCLE 34 ON FREE INFORMATION CARD

can set *passpoints,* breakpoints that will occur only when an instruction has been executed a number of times (ranging from 1 to 65535). In addition, you can *tag* an instruction or a group of instructions with a number ranging from 1 to 99; you can then set, clear, or disable breakpoints by tag number. The ML debugger also provides DEGUG-like facilities for searching, filling, moving, and disassembling memory, computing hex values, inputting and outputting values to I/O ports, etc. You can also append comments to disassembled instructions, but the comments may not be saved. A macro facility allows you to record sequences of keystrokes and assign them to any key. Macros may not be saved. TSM will also accept keystrokes from a file.

In the machine-code window, you can use a non-symbolic assembler to enter assembly-language instructions into your program in one of two ways. First, by pressing Ctrl-A, you enter an overwrite mode in which your instructions overwrite whatever instructions currently reside in memory. Second, by pressing Ctrl-P, you enter a patch mode in which your instructions are inserted *between* current instructions—even if there is no space for them!

You can also patch memory in the memory-dump window, in either hex or ASCII notation. You can also force the memory-dump window to track the machine-code window, so that any time you execute a machine instruction that accesses memory, that area of memory will be displayed.

One deficiency is that you can't use TSM to debug graphics programs. Future versions of the program may correct that deficiency, but for now, you'll have to stick to text-mode debugging. Our only other complaint is that window position is hard to control; by contrast, some machine-language debuggers allow window placement by screen line number.

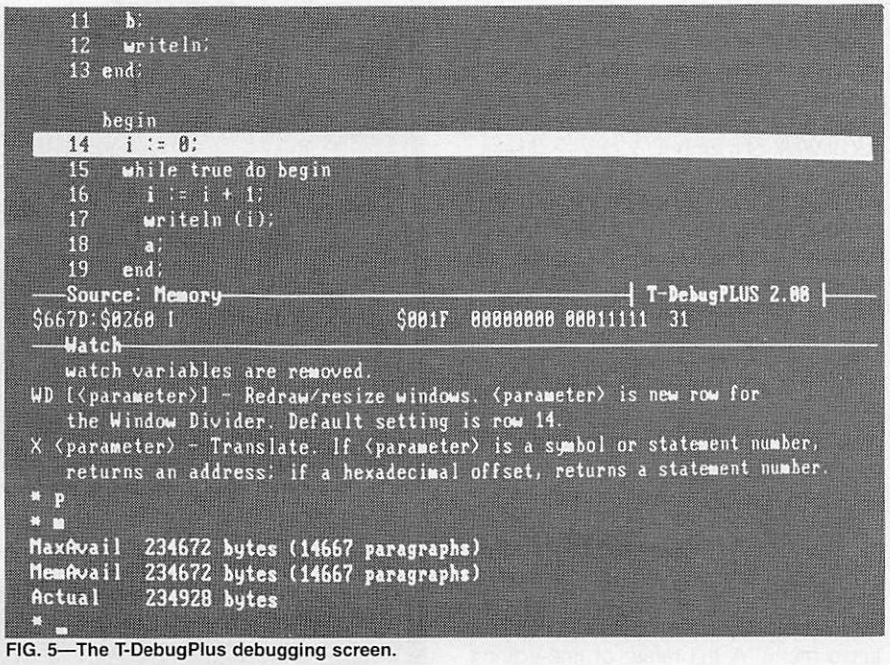


FIG. 5—The T-DebugPlus debugging screen.

T-DebugPlus

Turbo Power has been selling Turbo enhancements for a long time, and the company has a number of products, including T-DebugPlus; Turbo Extender, which allows you to create very large (greater than 64K) Turbo programs; Turbo Optimizer, which allows you to compact the size of run-time files and to create and link libraries of often-used routines; and Programmer's Utilities, which contains a program structure analyzer, an execution timer and profiler, pretty printer, and several file-management utilities. Source code is available for all programs, all of which are written in Turbo Pascal.

T-DebugPlus is contained in one .COM file and one overlay file, which together occupy about 100K of disk space. To run T-DebugPlus you need about 256K of free memory. The main features that distinguish T-DebugPlus from TSM are that (a) the program is written in Turbo Pascal, and the source is included; (b) the program has no built-in machine-language debugger; (c) the program does support graphics modes; (d) program and debug screens can appear on either mono or color monitors in a dual-monitor system. T-DebugPlus also has a screen-swapping mode.

In addition, T-DebugPlus includes several useful utilities, including TMAP, which generates a .MAP file for use with an external symbolic machine-language debugger; TMERGE, which allows T-DebugPlus to debug large Turbo programs (when used in conjunction with Turbo Extender), and two program-listing programs, one of which generates a symbolic disassembly. In addition, several files contain information about various routines in Turbo's run-time library; that information is extremely useful for learning about now Turbo works. All source files are contained in a single .ARC file on the distribution disk.

To use T-DebugPlus, you run the program, which, like TurboSmith, loads and runs Turbo for you. When you Run your program from Turbo's menu, T-DebugPlus's debugging screen comes up. (See Fig. 5). There you can single-step individual statements and entire functions and procedures. You can set breakpoints that become active when a variable changes or becomes equal to a specified value, or when a particular memory location is altered. You can also set breakpoints by line number and by routine name. A breakpoint may also include passpoint count values, so that the breakpoint would be executed after a routine was executed the specified number of times. In addition, you can open a memory-dump window in one of several formats, display "watch" variables, whose values are updated on the screen each time a breakpoint is reached, and examine and change the values of variables.

All commands are executed from a command line, which can be edited. No macros are allowed, but the function keys provide shortcuts to the most common commands, and F4 repeats the last command.

Although T-DebugPlus has no built-in machine-language interface, it can be used in conjunction with an external debugger—even Debug. The DG command will drop you into your debugger. Source-level debugging is not possible in that manner. However, you can use TMAP, one of the utilities included with the package, to generate a .MAP file, which most source-level debuggers can use to link source code and machine instructions.

Several miscellaneous commands provide useful functions. For example, a brief command summary is available by pressing F1. In addition, the X command allows you to find the machine addresses of variables, source-code line numbers, procedures, and functions. Conversely, you can find the nearest statement to a specified address.

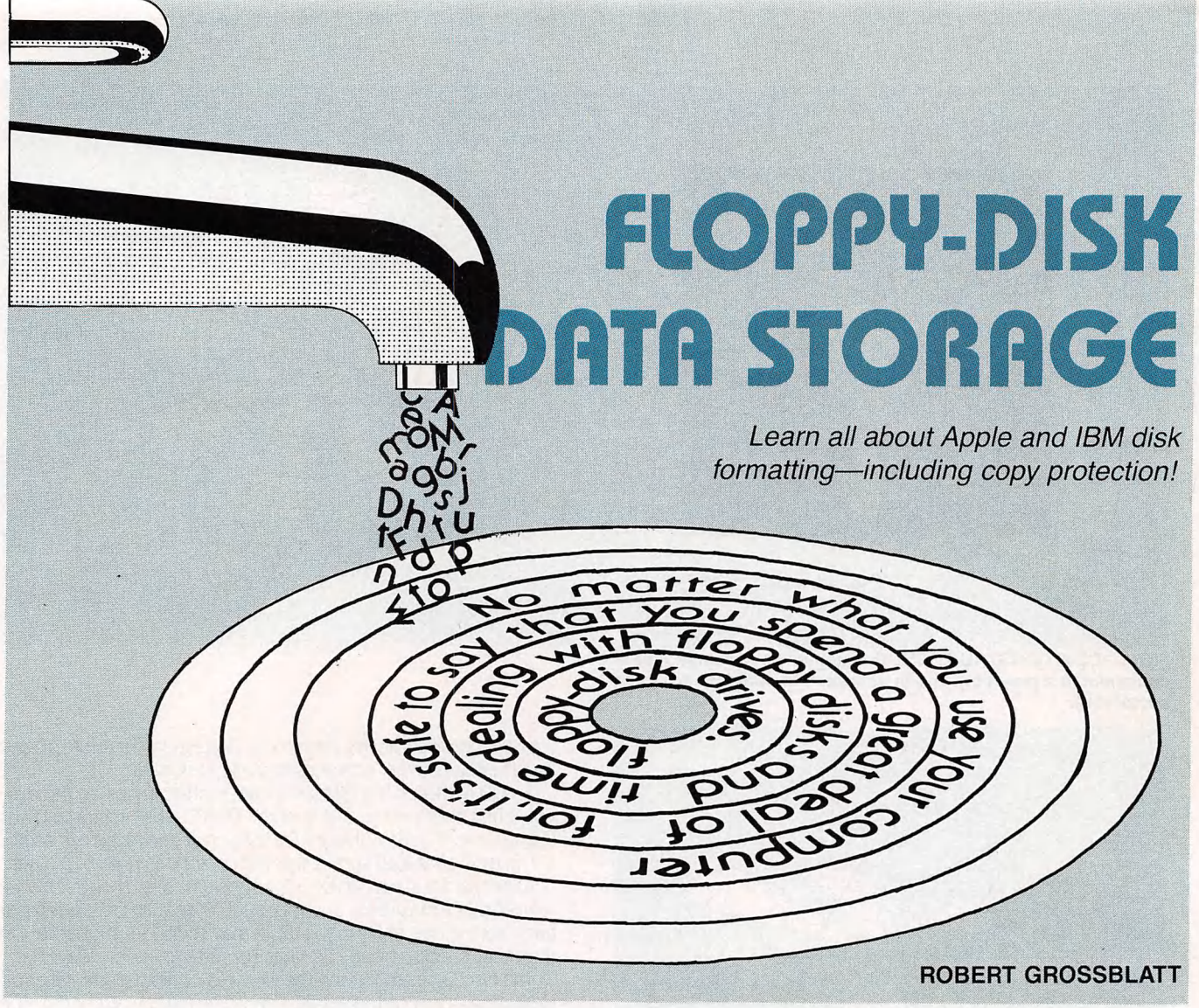
Conclusions

T-DebugPlus and TurboSmith are both extremely useful accessories to anyone programming in Turbo Pascal.

Points to keep in mind, however, are that T-DebugPlus does graphics, has full dual-monitor support, and slightly more convenient breakpoint facilities (at the source level). In addition, it requires less memory, and its documentation is more concise and better produced. For beginners and those not needing full machine-language support, it may be the better choice. TurboSmith's strength are its windows, its built-in machine-language interface, and its ability to work with a break-out switch. But considering the reasonable prices of these packages, you may want to get copies of each.

FLOPPY-DISK DATA STORAGE

Learn all about Apple and IBM disk formatting—including copy protection!


No matter what you use your computer for, it's safe to say that you spend a great deal of time dealing with floppy disks and floppy-disk drives.

ROBERT GROSSBLATT

No matter what you use your computer for, it's safe to say that you spend a great deal of time dealing with floppy disks and floppy-disk drives. Loading programs and saving data are such common operations that we tend to forget how fragile the whole system is. But all it takes is one disk disaster to remind us of that fragility.

Of course, there are ways of protecting against those types of disasters, and other ways of dealing with them when they do occur. Performing regular backups is the best protection, but even that is not fail-safe. What happens if a disk crashes during a backup procedure?

In order to have any chance at all of recovering that data, as well as to back up copy-protected software, you need to know how data is stored on your disks. The more of the process you understand, the better your chances of successfully recovering a crashed disk. So in this article we'll examine how data is stored on both IBM and Apple floppy disks. The information provided will put you far on the road toward being a real "disk jockey."

Tracks and sectors

The standard 5¼-inch floppy disk consists of a disk of magnetically coated plastic that is contained in a jacket, as shown in Fig. 1-a. In order for your computer to use the disk, it must have a way of finding its way around the magnetic coating on the surface. It does so by treating the disk as a group of tracks that are divided into sectors. As shown in Fig. 1-b, the tracks are a series of concentric circles, each of which is divided into a number of segments, the sectors. In addition to tracks and sectors, disks also have two sides, as shown in Fig. 2. Not all disk control hardware and software can use both sides, however.

The number of tracks and sectors determines how much data will fit on the disk. That amount is dependent on your computer's hardware and disk operating system (DOS). The numbers vary among computers and disk sizes, but the basic principles of operation are the same.

When you tell your computer to format a disk, the hardware moves the read/write head to track zero, the outermost track, and then forces it to deposit information on the surface of the disk that indicates the sector locations. The process is repeated for each track until the last track has been formatted.

Standard 5¼-inch Apple disks have 35 tracks on one side of the disk only, and the most common IBM format has 40 tracks on each side of the disk. Double-sided 3½-inch and 8-inch disks have 77 tracks per side, and the AT's quad-density 5¼-inch disks have 80 tracks on each side.

DOS (IBM or Apple) uses tracks and sectors to organize the disk's surface. At the DOS level, to find a particular piece of information, all you need are two pieces of information: track and sector numbers. With double-sided disks, you must also specify the head number.

The number of tracks per disk is usually a function of the hardware. The DOS talks to the disk controller, which, in turn, talks to the stepper motor in the drive and tells it to move the head in or out the desired number of tracks.

The number of sectors, however, is controlled by the DOS. IBM's DOS, for example, can format for eight or nine sectors per track, but standard Apple disks have sixteen sectors per track. So you can have more small sectors or fewer large sectors.

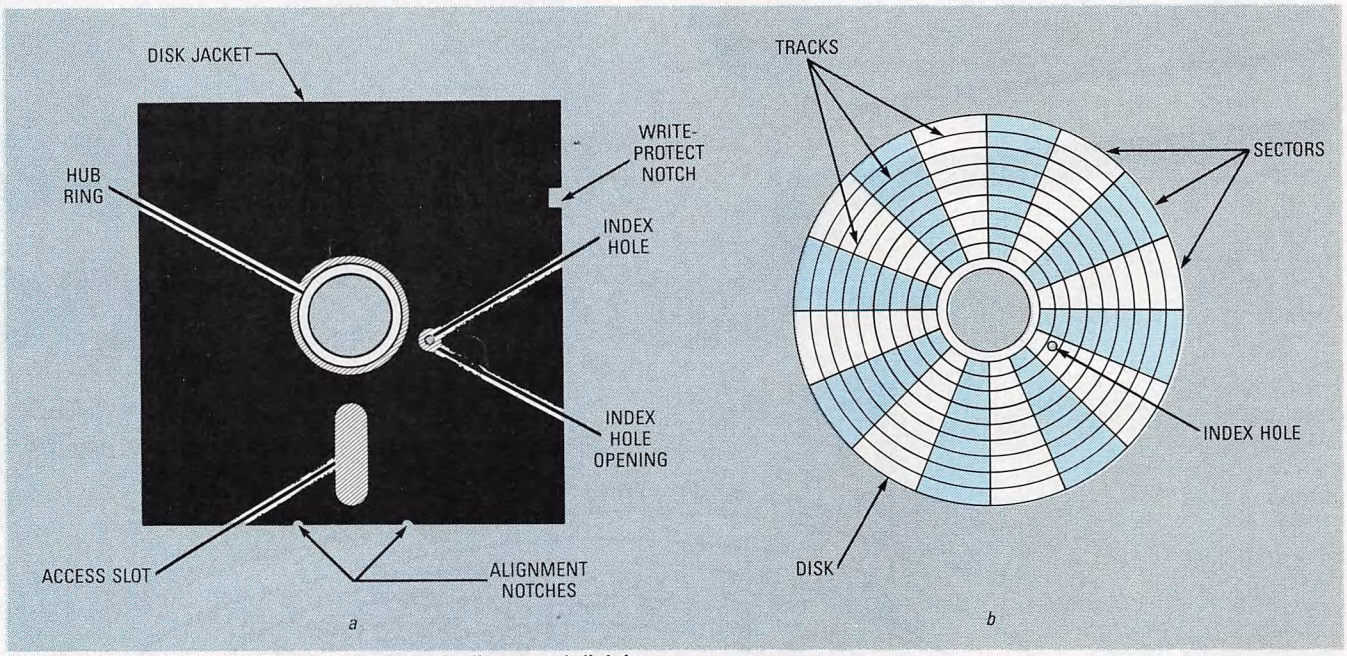


FIG. 1—DISK CONSTRUCTION: The magnetically coated disk is contained in a jacket (a), and is formatted to contain tracks and sectors (b).

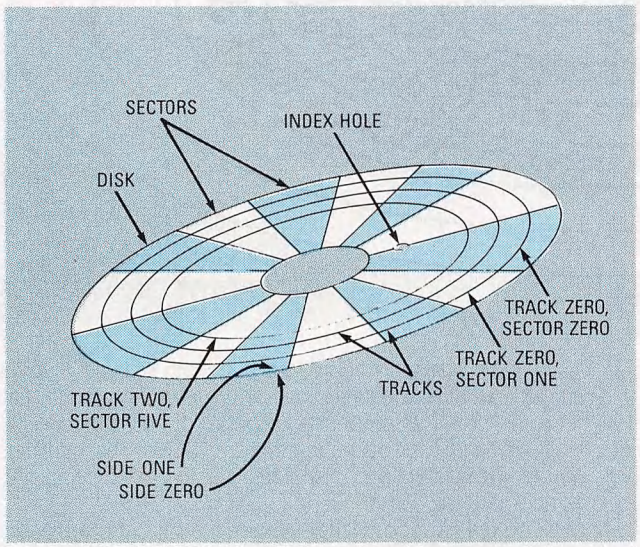


FIG. 2—BOTH SIDES of a disk are used by some disk control hardware and software.

Disk formatting

When a track is formatted, DOS writes three kinds of information in each sector: ID bytes, sync bytes, and gap bytes. The exact format of those bytes differs from computer to computer, but the same sort of scheme is used by every DOS. The reason is that DOS must have a way of determining exactly which sector it's looking at. Not only that, but there must be a way of ensuring that the special formatting bytes are never overwritten by data. If that does happen, DOS has no way to identify the sector and the result is what you might expect—a crashed disk.

There are actually two kinds of ID bytes on a sector—one is the signpost that marks the sector's location on the disk, and the other lets DOS know that it's looking at the beginning of the data stored in the sector.

Figure 3-a shows a dump of an Apple DOS 3.3 sector, and Fig. 3-b shows a dump from an IBM DOS 3.1 sector. At first glance, they both look meaningless—clearly different but equally meaningless. Those disk formats are the two most popular, and both the hardware and the software used to create them are totally incompatible. It's even more interesting, therefore, to see that they use similar schemes to write disk data.

The ID marks on the IBM sector are written in hex on the disk; you'll find them in Fig. 3-b at offset 00A1h. The first three bytes (00, 00, and 01) show that you're looking at track 0, side 0, sector 1. The next byte (02) shows that the sector can hold 512 bytes.

Other sector sizes can be accommodated, as shown in Table 1. Normally, a maximum of about 6000 bytes can be written per track, so the final entry in the table may seem questionable. On the other hand, perhaps IBM has something up its sleeve.

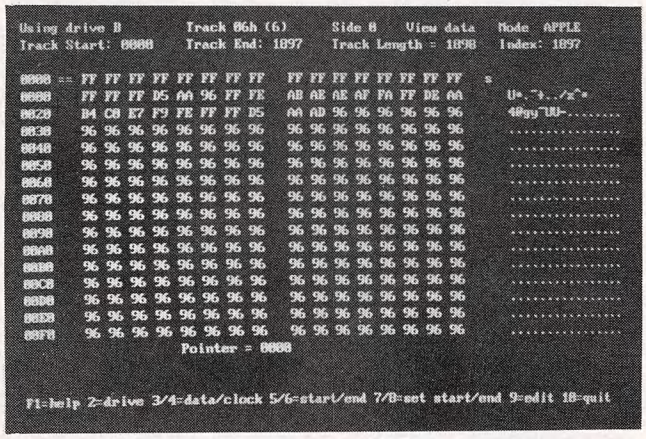
The two bytes following the ID bytes contain a special error-detecting code called a CRC (cyclic redundancy check). The CRC is used by DOS to make sure that data read from the disk is correct. If the CRC calculated from the data that is read from the disk doesn't match the four CRC bytes in the header, DOS considers the data corrupt. Every time you change the data in a sector, DOS recalculates the CRC and writes it to the disk.

In order to keep those bytes from being overwritten accidentally, DOS uses sync bytes to mark the location of the ID bytes. When the floppy-disk controller writes a data byte on the disk, it sends out a steady clocked stream of ones and zeros. The Apple, for example, writes bytes to the disk at intervals of 32 microseconds. Sync bytes, however, are written at a different interval so they're easy to spot on the disk. Apple sync bytes are written in 40-microsecond intervals, and each sync "byte" is 10 bits long.

IBM sync bytes differ. The IBM sector in Fig. 3-b shows that there are three bytes containing a value of A1 beginning at offset 9D. Those are the specially written sync bytes that the floppy-disk controller uses to mark the location of the ID bytes. The twelve 00 bytes preceding the A1 bytes are also sync bytes. You can understand how they're used by tracing through the mechanics of a normal disk read.

When an IBM controller must read data, the first thing it does is make sure that it's looking at the right sector. It starts reading data, watching for a stream of 00 sync bytes, which lets it know that there's a chance that A1 sync bytes will follow. If they do, DOS knows that the following bytes are ID bytes.

Although ID bytes are used to mark both the signposts and your data, DOS can tell the difference by looking at the byte immediately following the A1 bytes. If it's an FE, the ID bytes are signposts, but if it's an FB then it's data. The amount of data is


Using drive B Track 06h (6) Side 0 View data Mode APPLE
Track Start: 0000 Track End: 1097 Track Length = 1098 Index: 1097
Pointer = 0000
F1-help 2-drive 3/4-data/clock 5/6-start/end 7/8-set start/end 9-edit 10-quit

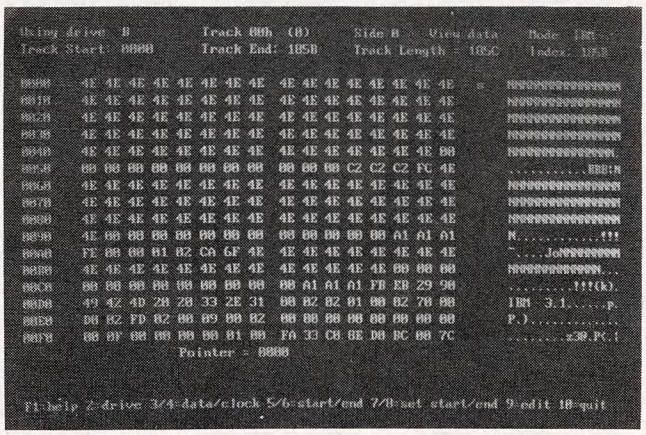

Using drive B Track 00h (0) Side 0 View data Mode IBM
Track Start: 0000 Track End: 1050 Track Length = 105C Index: 1051
Pointer = 0000
F1-help 2-drive 3/4-data/clock 5/6-start/end 7/8-set start/end 9-edit 10-quit

FIG. 3—SECTOR DUMP of an Apple disk (a) and an IBM disk (b).

known because the sector size is specified in the signpost.

The last non-data byte on the disk is called a gap byte. Gap bytes are insurance against worst-case operation. They're needed because not all disk drives turn at the same speed, so there's no way to guarantee that writing a new block of data to a sector won't overwrite existing ID and sync bytes. A disk drive only has one head per surface, so there's no way to read and write simultaneously. As long as drive speed is within tolerance, the DOS standards have been set so that there's no possibility of destroying any of the critical bytes needed to read the sector. On an IBM disk, the gap bytes usually have a value of 4E. Apple, on the other hand, uses 10-bit FF "bytes."

As for data bytes, if the sector hasn't been used, it will be filled with the DOS formatting bytes: IBM uses F6, Apple uses 96, and CP/M uses E5.

TABLE 1—IBM SECTOR SIZE ENCODING

ID Byte	Bytes/Sector
$00	128
$ 01	256
$02	512
$03	1024
$04	2048
$05	4096
$06	8192

TABLE 2—APPLE DISK ENCODING

One Byte Volume Number	=	B7 B6 B5 B4 B3 B2 B1 B0
Two Byte Disk Encoding	=	$FF FE
	=	11 11 11 11 11 11 11 10
Apple's Encoded Format	=	1B7 1B5 1B3 1B1 1B6 1B4 1B2 1B0
Decoded Binary Number	=	1 1 1 1 1 1 1 0
Volume Number	=	$FE
	=	254
One Byte Track Number	=	B7 B6 B5 B4 B3 B2 B1 B0
Two Byte Disk Encoding	=	$AB AE
	=	10 10 10 11 10 10 11 10
Apple's Encoded Format	=	1B7 1B5 1B3 1B1 1B6 1B4 1B2 1B0
Decoded Binary Number	=	0 0 0 0 0 1 1 0
Track Number	=	$06
	=	6
One Byte Sector Number	=	B7 B6 B5 B4 B3 B2 B1 B0
Two Byte Disk Encoding	=	$AE AF
	=	10 10 11 10 10 10 11 11
Apple's Encoded Format	=	1B7 1B5 1B3 1B1 1B6 1B4 1B2 1B0
Decoded Binary Number	=	0 0 0 0 1 1 0 1
Sector Number	=	$OD
	=	13

Although Apple's disk format is structurally similar to IBM's, the details are different because Apple's disk control hardware and software are unique. Most disk controllers store data in unencoded format, so that a dump of an ASCII text file, for example, will be comprehensible.

The Apple hardware, however, limits the values that can be stored on disk. The high bit of each byte must be set, there can't be more than two adjacent zero bits, and at least two adjacent bits must be set in each byte. Some values are reserved for use as ID bytes, and hardware restrictions eliminate many others, so there are only 64 possible values that can be written to the disk to represent your data.

So, in order to be able to write all 256 combinations of eight bits, it's clear that the data must be encoded. In fact, Apple has gone through three major revisions of their encoding scheme. However, that's not a subject that can be covered here; see the books in the References sidebar for more information.

Apple format

Apple's sector format is somewhat different. Referring back to Fig. 3-a, the signpost ID bytes are located at offset 0013h after the series of FF sync bytes. The signpost bytes always begin with a prologue (D5 AA 96), which serves the same purpose as the FE marking the IBM signpost. The next eight bytes are encoded versions of the disk volume number, track, sector, and checksum. As shown in Table 2, by decoding them we see we're looking at volume 254, track 6, sector 13. The checksum is calculated by sequentially XORing all data bytes in that sector together.

Following that information is an epilogue, which can be seen beginning at offset 001Eh. It marks the end of the signpost area and has no counterpart on an IBM disk. The epilogue is there so that DOS can make sure it's been reading the correct signpost marks and that it is still in sync with the disk. They're not really necessary, but remember that the Apple system was devised in the late seventies when disk drives were not as reliable as they are today.

Following another group of sync bytes comes the data bytes. At offset 0028h is the prologue (D5 AA AD); then follow 342 bytes of data. Apple stores 256 bytes of data in each sector, but, because the data is encoded, 342 bytes are needed to do it. A checksum is calculated for the data and stored at the end of the data space along with the epilogue (DE AA EB).

Sector numbering

Although sectors are numbered sequentially, often they're not stored sequentially. When DOS looks for a particular sector, it must locate the signpost markers, read them, verify the read, and then see if they're the ones it was looking for. All that takes time, but meanwhile—the disk keeps spinning, so there's a good chance the next sector will have passed beneath the read/write head while the previous sector was being analyzed.

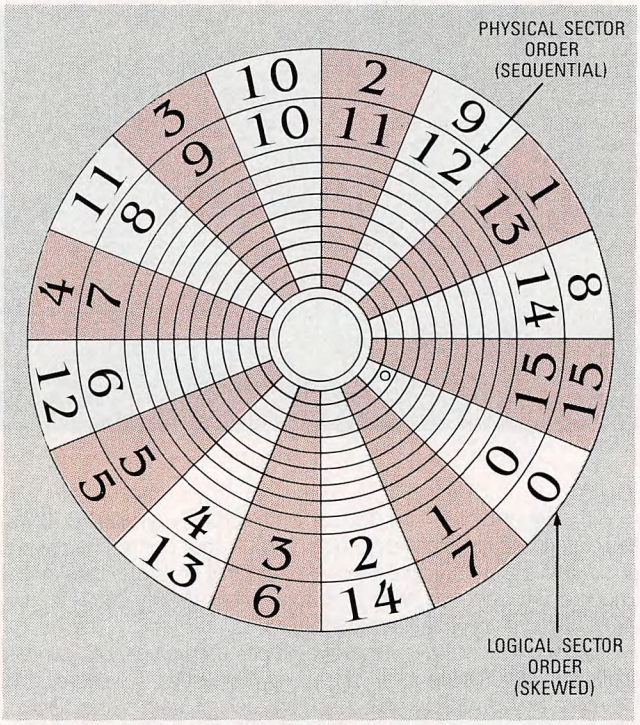


FIG. 4—LOGICAL AND PHYSICAL sector orders are not necessarily the same.

So, to make things more efficient, the sectors are interleaved, or skewed, as shown in Fig. 4. The inner circle indicates physical sector numbering; the outer circle indicates logical sector numbering.

Disk organization

Now that we know how data is stored on the disk, the next step is to see how it's organized. Once again, although the details vary from computer to computer, the basic method is the same. Let's look at the general principles and then see how they're actually applied in both Apple and IBM systems.

DOS divides the disk into four areas: System, Directory, Table of Contents, and Data. Any disk operation makes use of the information stored in all four areas.

DOS itself is stored in the System area of the disk—the first few tracks on the disk. The number of tracks depends on the size of DOS and the type of computer. The very first sector on the first track is called the *boot sector* and it's used whenever you boot a disk. It has a short machine-language program that tells the computer how to go about loading DOS from the disk. In addition, in some systems it contains data related to the directory structure, disk format, and so on.

When you boot your computer, the disk controller reads the boot sector into memory. Then the computer turns control over to the program, called the bootstrap loader, that is contained in that sector. Things are done that way because the controller can read in only a sector at a time. You can read in more than one sector at a time—but that's DOS's job. So it's a chicken-and-egg problem: you need DOS to read multiple sectors and you can only read in multiple sectors by using DOS. But because DOS is located in a specific place on the disk, the bootstrap loader knows how to transfer it to the computer's memory.

The Directory and the Table of Contents go hand in hand. The former is a list of the files stored on the disk, and the latter is a list of sectors telling DOS where to find them. Every time you tell DOS to access a particular file, it first goes to the Directory to see if the file exists and, if it does, DOS then turns to the Table of Contents to get the file's location on the disk.

As you might have guessed, the Data space is where DOS stores your data.

IBM stores DOS in three disk files: IBMBIO.COM, IBMDOS.COM, and COMMAND.COM. (Clones running MS-DOS store the first two of those programs under different names.) In order for the computer to load DOS from the disk, IBMBIO.COM and IBMDOS.COM must be the first two files on the disk. If they're not, the bootstrap loader won't be able to find them, and you'll get the infamous "NON-SYSTEM DISK OR DISK ERROR" message. Those two files contain the routines the computer uses to control the disk hardware. The third file, COMMAND.COM, is loaded after the first two and it's the part of DOS that executes internal DOS commands (DIR, REN, DEL, etc.), external programs, and batch files.

IBM's Table of Contents is called the FAT (file allocation table). It's used to keep track of where each chunk of each file is stored on the disk, which sectors are available for use, which sectors are bad, and so on. The basic unit of the FAT is called the cluster, and it represents a variable number of disk sectors, depending on the operating system and on the format of the disk. For example, single-sided IBM disk clusters are one sector long, double-sided IBM disk clusters are two sectors long, and higher density disks have even larger clusters.

The Directory is also located on track 0, right after the FAT. It stores the names of your files, their attributes, the date and time they were created, their size, and the location of the file's first entry in the FAT. The rest of the disk is set aside for your data. When you change anything on the disk by writing new data, DOS must update the Directory and the FAT, because they work together to keep your files organized.

The Apple also has a boot sector to start the process of loading DOS into the computer. The actual contents of the boot sector (as well as the organization of the disk) depends on which Apple DOS you're using. Both systems, however, differ from the IBM version in that the only job of Apple's boot sector is to start the process of loading in DOS.

There are major differences in the two current Apple operating systems, DOS 3.3 and ProDOS. Although the basic sector formatting is the same, the organization of the disk is quite different. The older system, DOS 3.3, stores the System on the first three tracks, and the directory on track 11h (in the middle of the disk). That was done, the reasoning went, because, on average, the head would have less distance to travel to get to the directory than if the directory had been located in track 0, as on the IBM. Less distance means less time, so that wasn't bad reasoning.

ProDOS, on the other hand, has a closer resemblance to the IBM system in that the directory is stored in the lower tracks. Disk space is allocated a sector at a time under DOS 3.3, and a block at a time under ProDOS. (A block equals two consecutive sectors.) The Table of Contents is called the Volume Table of Contents in DOS 3.3 and the Volume Bit Map in ProDOS.

Both are similar to the IBM's FAT in that each contains a table that DOS uses to keep track of which sectors are free, which are reserved, and which are otherwise used. Each file on the disk reserves a track-and-sector list sector (or block) that contains a list of the sectors where each file's data is stored. Overall, the Apple has to do about the same amount of housekeeping as the IBM. It must update the directory, the table of contents, and the track/sector list for each file you use.

Copy protection

With those facts in mind, let's examine various copy-protection schemes. But before we get into the details, let's talk about the philosophy behind copy protection. Both software publishers and software owners make strong arguments about protecting their investment—and they're both right. Nobody wants to get ripped off, so publishers should be able to make money, and owners should be able to make legitimate backup copies of software they have purchased. Of course, there is much

continued on page 100

BUILD THE PT-68K

The big moment: add the microprocessor.

©1987, Peter A. Stark

PETER STARK, STARK SOFTWARE SYSTEMS CORPORATION

Part 3 In the first installment of the **Computer Digest** Classroom, we presented an overview of the PT-68K. In part two, we built clock, reset, and test circuits. Now let's install the 68000 and start learning how it works.

Step 7: hardware basics

The pinout of the 68000 microprocessor is shown in Fig. 1. Though it's a 64-pin IC and looks complex, it really is straightforward. Let's go over it pin by pin.

In the figure, notice first the data bus, with its 16 lines labeled D0–D15, and the address bus, with its 23 lines labeled A1–A23. In case you're wondering, there is no A0—see below.

The remaining signals are known collectively as the control lines; let's look at them in more detail. Each control line is labeled with an arrow to indicate whether it is an input, an output, or, in some cases, both.

The three active-high output lines FC0, FC1, and FC2 output a function code that can be decoded to indicate what the 68000 is doing internally; the function code can also be used to increase the 68000's addressable memory to 64 megabytes.

The enable line (E), the valid memory address line (VMA), and the valid peripheral address (VPA) line are all useful when the 68000 is used with older input-output IC's, particularly those originally intended for use with Motorola's 6800 processor. Also, VPA provides some interrupt information.

IPL0, IPL1, and IPL2 are interrupt-level inputs. We will discuss interrupts later; for now let us just say that an event such as a keypress can interrupt the 68000, cause it to stop whatever it's doing, and then respond to the interrupt. The three interrupt inputs tell the 68000 whether an interrupt is being asked for, and what kind of an interrupt it is.

The RESET and HALT inputs come from the 555 circuit shown in Part 2, Fig. 6. Note, however, that those two pins are also outputs. That explains why an open-collector 7406 inverter was used to drive them; occasionally the 68000 may output a low on one of those lines, and that would conflict with the normally high output of a standard inverter (such as a 7404).

BR, BG, and BGACK are used with DMA circuitry, which is used to transfer blocks of data without help from the microprocessor. If DMA were used, the DMA controller would send a bus request (BR) signal to the 68000, which would release the data and

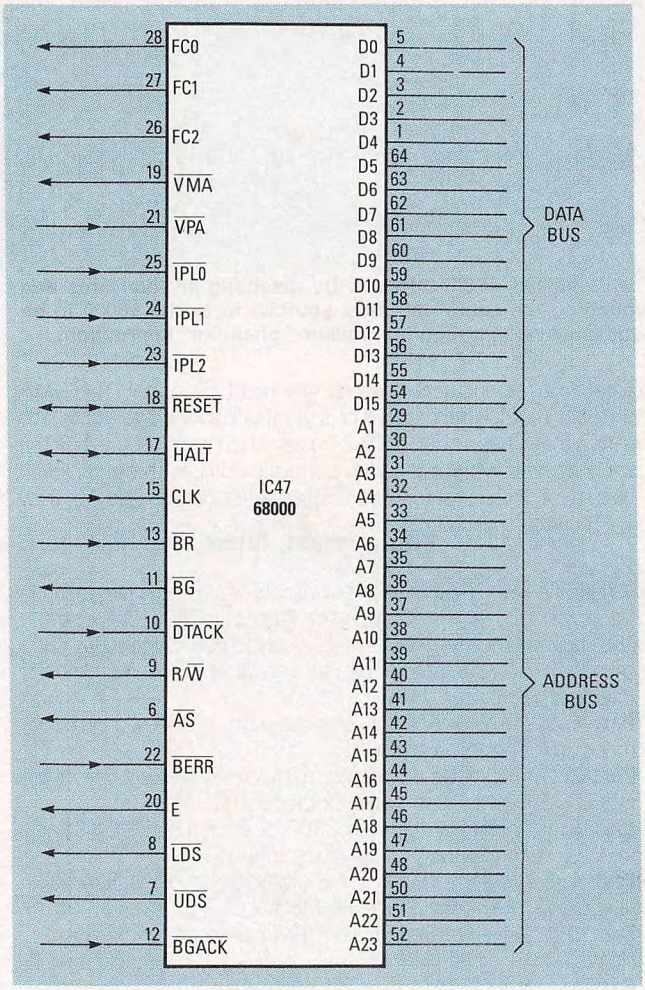


FIG. 1—THE 68000 MICROPROCESSOR has sixteen data lines, 23 address lines, and 21 control lines.

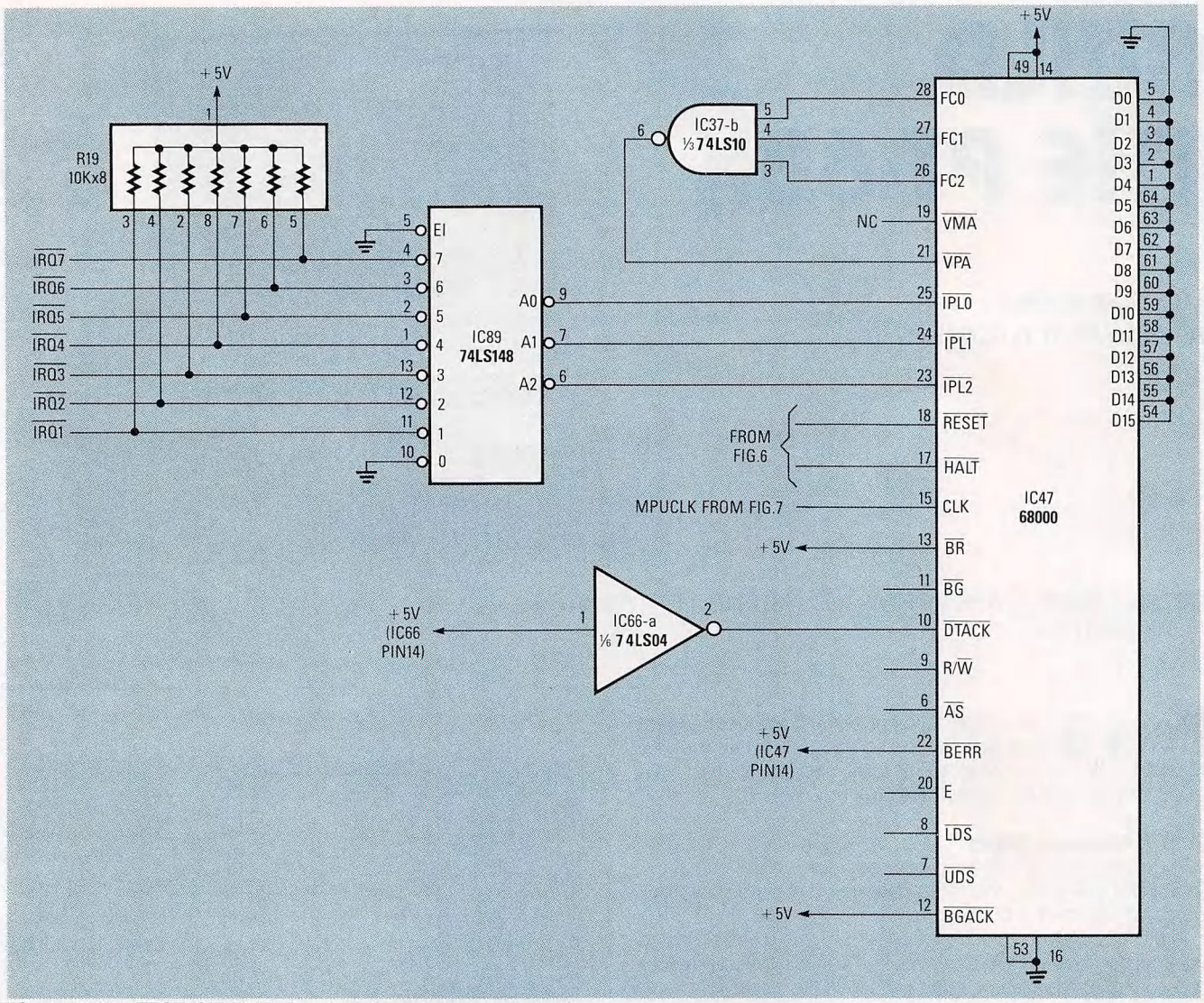


FIG. 2—68000 TEST CIRCUIT. By disabling all interrupts and grounding the entire data bus, you can force the 68000 to repetitively cycle through four million "phantom" instructions.

address buses and return a bus granted (BG) signal. The DMA controller would then send a bus grant acknowledge (BGACK) to confirm that it has control of the buses. Then the 68000 would sit back and wait while the DMA controller did its thing.

We mentioned that LDS and UDS replace address line A0; they do so in an interesting way. The 68000 has a 16-bit data bus, but memory is organized as eight-bit bytes. Even so, the data bus can access two bytes at a time. The memory is wired so that half of memory—the odd-numbered locations—connects to the lower part of the data bus (bits D0–D7), and the other half of memory—the even-numbered locations—connects to the upper part of the data bus (bits D8–D15). The 68000 asserts LDS when it wants to use the lower half of the data bus, UDS if it wants to use the upper half of the data bus, or both if it wants to transfer 16 bits on the entire data bus. Thus an odd address turns on LDS, and an even address turns on UDS. The overall effect is similar to that provided by address line A0 in other microprocessors, where A0 is low for an even address and high for an odd address.

AS is an address strobe which is generally asserted by the 68000 at the same time as either LDS or UDS; it simply tells external circuitry (address decoders, for example) that there is a valid address on the address bus. That's an important point, because the address bus often carries data that is meaningless; AS provides a way of preventing decoders from responding to invalid addresses.

Next comes the read/write line (R/W), which is used by the 68000 to tell other circuitry whether it wants to read data in (when R/W is high) or write data out (when R/W is low). In other words, R/W is high when data goes from RAM or ROM to the

Past, present, future

What follows is a listing of the contents of past and projected future articles in the **Computer Digest** 68000 Classroom. The precise number of articles—and their contents—will depend on your response, so *let us know what you're interested in!*

Part 1: System overview, block diagram, parts list, memory map, ordering information.

Part 2: Parts-placement diagram, power connector mounting, LED, speaker, reset, and clock circuits.

Part 3: Introduction to the 68000, test circuit, EPROM and RAM circuits, address-bus waveforms.

Part 4: Logic symbols, MAP circuit, address decoding, BERR and DTACK, RAM and ROM, HUMBUG.

Part 5: Serial interfacing and IBM-compatible expansion slots.

Part 6: Dynamic RAM.

Part 7: Disk control hardware and software. SK*DOS.

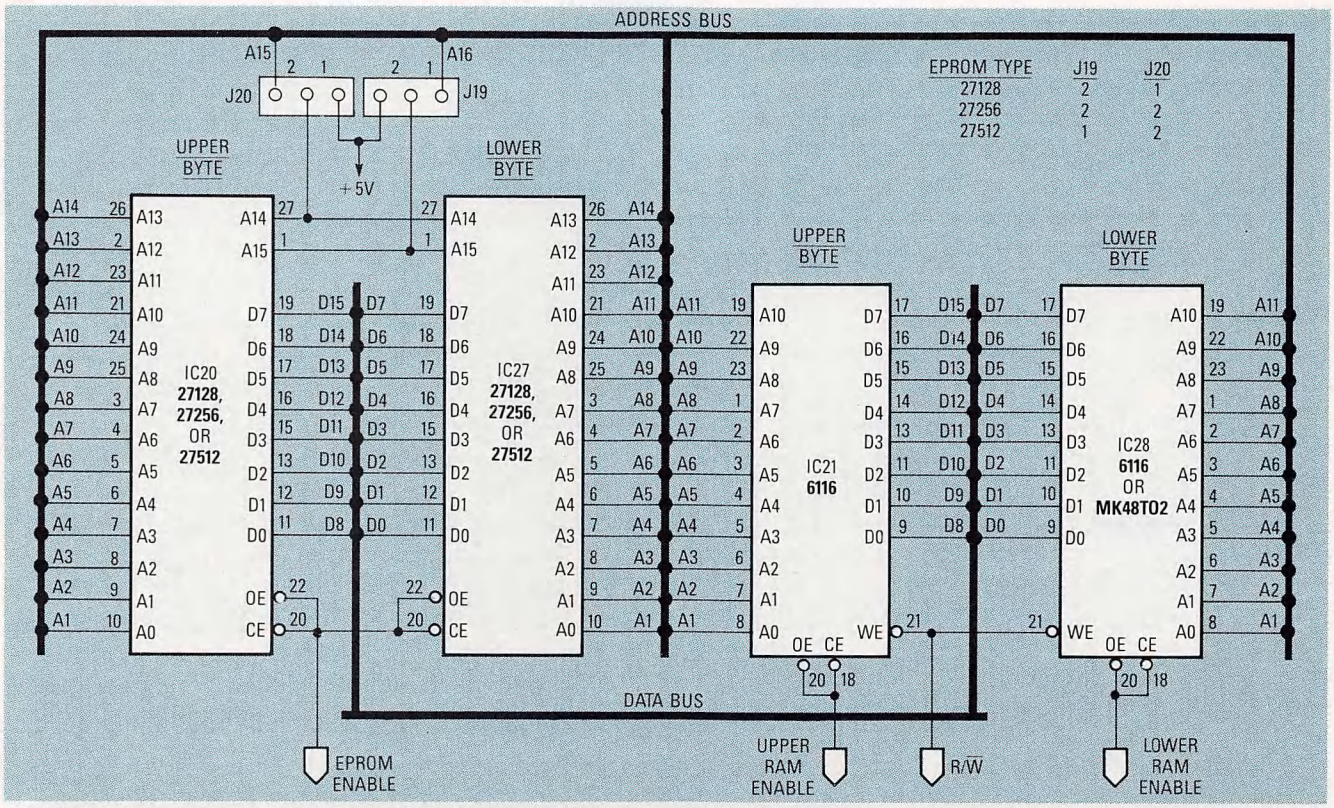


FIG. 3—EPROM AND STATIC-RAM WIRING is shown here.

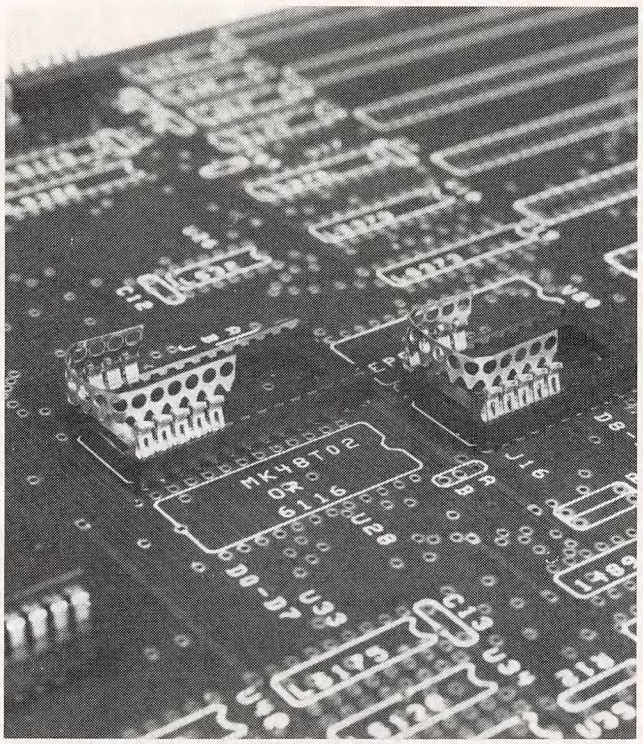


FIG. 4—USE MOLEX SOLDERCON PINS to ground all data lines without soldering.

Thanks

I'd like to thank Fred Brown of Peripheral Technology, Inc., kit supplier for this CD Classroom series. Fred is the wizard who designed the hardware of the PT-68K; without his assistance, the project would never have gotten off the ground.

68000, and low when data goes from the 68000 to RAM. Normally we don't write to a ROM.

$\overline{\text{BERR}}$ is an input to the 68000 that external circuitry uses to tell the 68000 when something has gone wrong on one of the buses. We will see how that is done later.

Last, $\overline{\text{DTACK}}$ stands for data transfer acknowledge. Whenever the 68000 wants to read or write to memory (or an I/O device), it (a) puts the address on the address bus, (b) puts a high or low on R/$\overline{\text{W}}$, (c) asserts $\overline{\text{AS}}$, (d) asserts $\overline{\text{LDS}}$, $\overline{\text{UDS}}$, or both, and (e) waits until either $\overline{\text{DTACK}}$ is asserted, indicating that the transfer has completed successfully, or $\overline{\text{BERR}}$ is asserted, indicating that something went wrong. When $\overline{\text{DTACK}}$ is received, the 68000 goes on to the next instruction. We'll discuss what happens if $\overline{\text{BERR}}$ is asserted later.

If $\overline{\text{DTACK}}$ were permanently grounded, the 68000 would assume that all transfers finished quickly, so it would zip along at maximum speed. In most cases, though, $\overline{\text{DTACK}}$ is generated by an external timer that gives memory and I/O just enough time to finish their jobs. And if a memory or I/O device is particularly slow, $\overline{\text{DTACK}}$ can then be delayed so that the 68000 will wait for it to finish.

In practice, each 68000 memory or I/O access takes a specific amount of time, which is measured in clock cycles. If $\overline{\text{DTACK}}$ is delayed, for even an instant, the 68000 lets an extra clock cycle slip by and checks again. If $\overline{\text{DTACK}}$ is still off, the 68000 waits another clock cycle, and so on. Each of those clock cycles is called a *wait state*. Ideally, everything would be fast enough so that the 68000 could continue processing without wait states. However, some computers have slow memory or I/O devices, and therefore run with one or even more wait states, which obviously slows everything down. You'll be happy to know that the PT-68K runs with no wait states!

68000 test circuit

Last time we built the clock and reset circuits, and now that we have some basic familiarity with the 68000, it's time to get the system running. Normally, you need quite a bit of external hardware to get a 68000-based computer running, but there is a

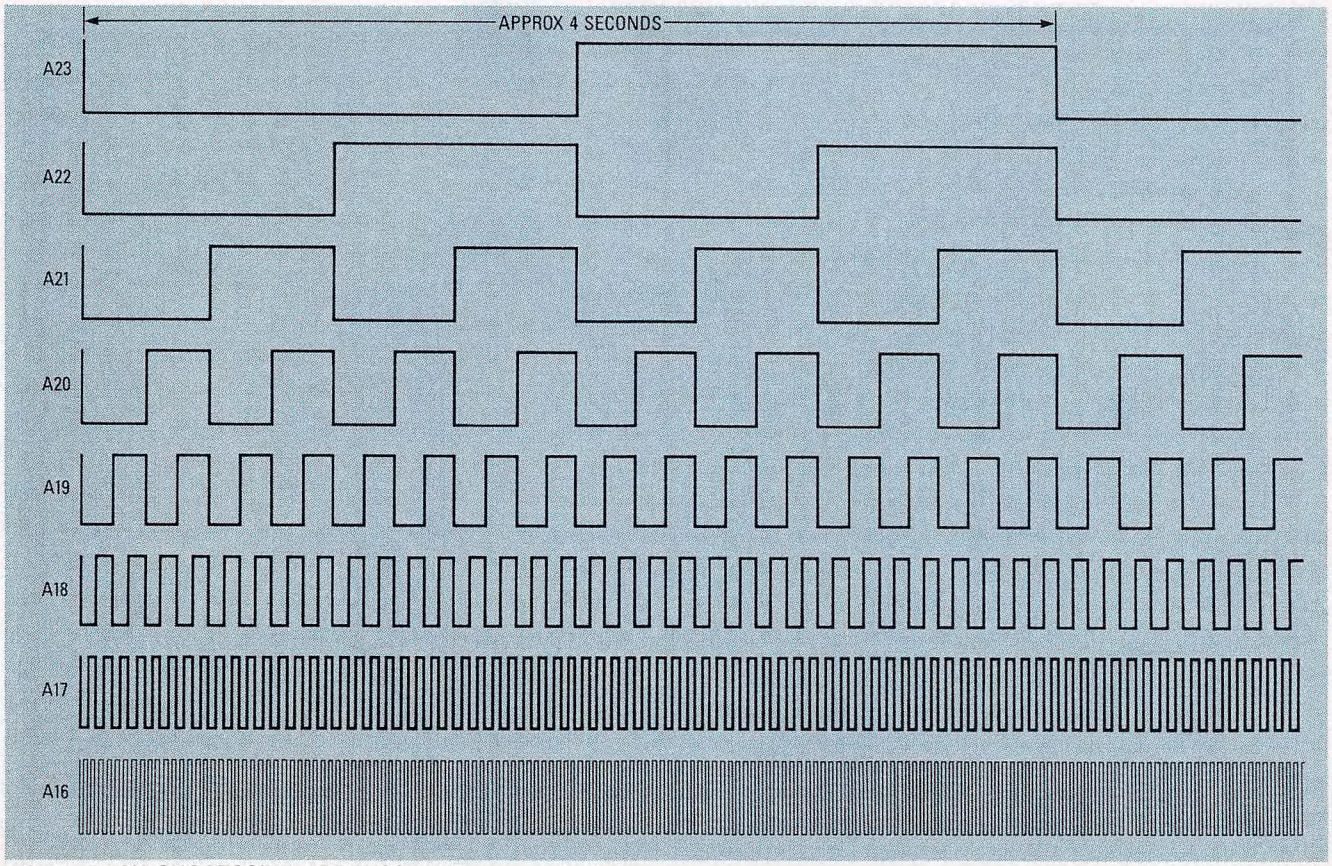


FIG. 5—EACH SUCCESSIVE ADDRESS LINE runs at half the frequency of its predecessor.

way of fooling the microprocessor into thinking the necessary support circuitry is connected, even though it isn't. Figure 2 shows how. Basically, the circuit ensures that interrupts are disabled and jams the data bus with "phantom" instructions that the 68000 will execute over and over.

In order to minimize the amount of extra wiring we have to do, we will take advantage of circuitry already on the printed-circuit board (which will be needed later anyway). For example, we can use $\overline{\text{RESET}}$, $\overline{\text{HALT}}$, and CLK as-is.

The circuit works like this: IC37-b is a NAND gate that asserts $\overline{\text{VPA}}$ when FC0, FC1, and FC2 are all high, and negates $\overline{\text{VPA}}$ at all other times. That is done because $\overline{\text{VPA}}$ is used only during interrupt processing; for now, we need to force it high.

Also dealing with interrupts is IC89, a priority-encoder IC. In normal operation, when an interrupt request appears on one of the $\overline{\text{IRQ}}$ lines ($\overline{\text{IRQ1}}$–$\overline{\text{IRQ7}}$), IC89 functions as a traffic cop that determines which line has the highest priority. $\overline{\text{IRQ7}}$ has a higher priority than $\overline{\text{IRQ6}}$, which is in turn higher than $\overline{\text{IRQ5}}$, and so on. Depending on which interrupt lines are active, IC89's three outputs send a binary number corresponding to the highest priority interrupt to the $\overline{\text{IPL0}}$, $\overline{\text{IPL1}}$, and $\overline{\text{IPL2}}$ lines of the 68000. For example, if the highest priority interrupt request is $\overline{\text{IRQ4}}$, U89 sends the binary number 100 (4) to the 68000 by forcing $\overline{\text{IPL2}}$ high and the others low.

In our test circuit, however, the seven resistors in R19 are pulling all of the $\overline{\text{IRQ}}$ lines high. Therefore, IC89 sees no interrupt request, so it sends the binary number 000 to the 68000, telling it that there are no interrupt requests.

By the way, we could have achieved the same result by grounding the three $\overline{\text{IPL}}$ pins of the 68000 and by tying $\overline{\text{VPA}}$ high. However, we can save ourselves extra labor by installing R19, IC89, and IC37, which we'll have to do eventually anyway.

Two of the inputs, labeled $\overline{\text{BR}}$ and $\overline{\text{BGACK}}$, are already tied high, and a number of other pins are unconnected.

That leaves the data bus, $\overline{\text{BERR}}$, and $\overline{\text{DTACK}}$ to contend with. First, $\overline{\text{BERR}}$ must be tied high so that the 68000 won't think a bus error has occurred. That is easily done by installing a short jumper between pins 14 and 22 of the 68000, on the foil side of the board.

Normally $\overline{\text{DTACK}}$ will carry a meaningful signal, but for now we want to ground it to make the 68000 think that all is well on the outside. The inverter (IC66-a) that drives $\overline{\text{DTACK}}$ was installed previously, so force the input high by installing a jumper from pin 1 to pin 14 of that IC on the foil side of the board.

Last, as shown in Fig. 2, we want to ground all sixteen data lines. The reason is that, when the 68000 is running normally, it fetches instructions and addresses from memory, so we have to provide it with some apparently meaningful data. By grounding the entire data bus, every time the 68000 tries to read anything from memory, it will read the number 0000. As it turns out, that is a valid 68000 machine-language instruction, which is written:

OR.B #0,D0

That instruction tells the microprocessor to *or* a 0 to register D0; the instruction consists of four 00 bytes.

Though that OR instruction seemingly does nothing, the 68000 thinks that all 16 megabytes of memory are filled with 4 million OR instructions, and so it starts executing them one after another. When it gets to the top of memory at $FFFFFF, it simply "wraps around" and starts over at $000000.

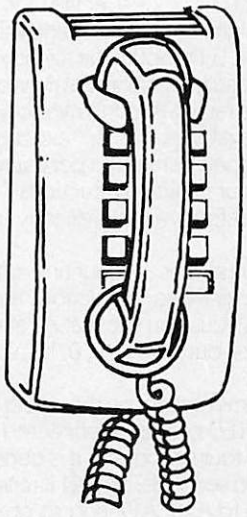
Fire it up

Now let's wire up the circuit and see what happens. Install the following components: sockets for IC37, IC47, and IC89; R19, a 10K single-in-line package. Pin 1 of R19, identified by a white line or dot, should point toward J25. Then install C14, C48, and C66 (0.1 μF disc capacitors), and last jumpers from pin 14 to pin 22 of IC47 and from pin 1 to pin 14 of IC66. Both jumpers will be removed later, so install them neatly and in a way such that they can be removed easily.

Grounding all sixteen lines of the data bus at the micro-

R-E Computer Admart

Rates: Ads are 2¼" × 2⅞". **One** insertion **$825. Six** insertions **$800** each. **Twelve** insertions **$775.** each. Closing date same as regular rate card. Send order with remittance to **Computer Admart,** Radio Electronics Magazine, 500-B Bi-County Blvd., Farmingdale, NY 11735. Direct telephone inquiries to Arline Fishman, area code-**516-293-3000.** ***Only 100% Computer ads are accepted for this Admart.***



CALL NOW AND RESERVE YOUR SPACE

- 6 × rate $800.00 per each insertion.
- Reaches 245,824 readers.
- Fast reader service cycle.
- Short lead time for the placement of ads.

Call 516-293-3000 to reserve space. Ask for Arline Fishman. Limited number of pages available. Mail materials to: Computer Admart, RADIO-ELECTRONICS, 500-B Bi-County Blvd., Farmingdale, NY 11735.

COMPUTER ASSEMBLY MANUALS

Eliminate Guesswork! Build with Confidence!

BIG BLUE SEED for IBM™ BUILDERS!
Parts list, placement diagrams, instructions, mods, for assembling over **90** IBM-compatible bare cards. Latest version with guides for 640K, Turbo, AT & Baby AT MthBds. **$17.95**

APPLE™ BUILDERS NEED APPLE SEED!
Instructions for assembling over **90** Apple-compatible bare cards including II+ & IIe MthBds. For ALL Apple enthusiasts and hobbyists **$14.95**

SAVE 10% - ORDER TWO OR MORE MANUALS OVER 40 BARE IBM & APPLE CARDS IN STOCK
(MthBds • Disk • Video • Burner • MultiFn • I/O • Modem • Proto • etc)

Check/Money-order, VISA/MasterCard to:
NuScope Associates*, Dept RE

In U.S.A.
P.O. Box 790
Lewiston, NY 14092

In CANADA
P.O. Box 742, Stn B
Willowdale, Ont. M2K 2R1

*A Division of Kosmic MicroTech Inc.

CIRCLE 194 ON FREE INFORMATION CARD

ICs PROMPT DELIVERY!!!
SAME DAY SHIPPING (USUALLY)
QUANTITY ONE PRICES SHOWN for SEPT. 27, 1987
OUTSIDE OKLAHOMA: NO SALES TAX

640K MOTHERBD UPGRADE: Zenith 150, IBM PC/XT, Compaq Portable & Plus; hp Vectra

DYNAMIC RAM				
1Mbit	1000Kx1	100 ns	$27.50	
1Mbit	256Kx4	120 ns	32.00	
51258	*256Kx1	80 ns	7.40	* STATIC COLUMN DRAM FOR 386 COMPAQ
4464	64Kx4	150 ns	3.60	
41256	256Kx1	80 ns	5.50	
41256	256Kx1	100 ns	4.85	
41256	256Kx1	120 ns	3.95	80387-16 $540.00
41256	256Kx1	150 ns	3.50	
41264	2-PORT	120 ns	5.25	
EPROM				
27C101	1000Kx1	150 ns	$45.00	
27C256	32Kx8	250 ns	6.75	80287-8 $250.00
27256	32Kx8	250 ns	5.60	
27128	16Kx8	250 ns	5.35	
STATIC RAM				
43256L-12	32Kx8	120 ns	$11.75	8087-2 $160.00
6264LP-15	8Kx8	150 ns	3.50	

OPEN 6½ DAYS, 7:30 AM-10 PM: SHIP VIA FED-EX ON SAT.
SUNDAYS & HOLIDAYS: SHIPMENT OR DELIVERY, VIA U.S. EXPRESS MAIL

SAT DELIVERY INCLUDED ON FED-EX ORDERS RECEIVED BY:
Th-Std Air 5A 1 lb
Fr. P-1 $10.50 2 lbs

MasterCard VISA or UPS CASH COD
Factory New, Prime Parts µP∞
MICROPROCESSORS UNLIMITED, INC.
24,000 S. Peoria Ave.
BEGGS, OK. 74421 **(918) 267-4961**
No minimum order. Please note that prices are subject to change. Shipping & insurance extra. A up to $1 for packing materials. **Orders received by 9 PM CST can usually be delivered the next morning, via Federal Express Standard Air** or $4.00, or guaranteed next day Priority One or $10.50! All parts guaranteed!

CIRCLE 61 ON FREE INFORMATION CARD

LASER PRINTER* P.C. COPIER OWNERS

*H.P., Apple, OASYS, QMS, and more.

Recharge Your Used Toner Cartridges.

RECHARGING:	PRINTER	COPIER
BLACK	$45	$40
DK BLUE/BROWN	$65	$60
NEW CARTRIDGE	$89	

(RETURN SHIPPING INCLUDED)

20 Years of Laser Printing and Xerography Experience -Unconditional Guarantee-

LASER EDGE

800 W. Burlington, Fairfield, IA 52556
(800)635-8088 (515)472-7850

CIRCLE 197 ON FREE INFORMATION CARD

COMPUTER PROGRAMS FOR Amway® PRODUCT DISTRIBUTORS*

IBM PC/COMPATIBLES

- TEN PROGRAMS
- 100% BASIC
- EASY TO LEARN
- EASY TO USE
- WRITTEN BY EMERALD DIRECT
- SINCE 1981

NOT COPY PROTECTED

CA RESIDENTS ADD STATE TAX **$49.95 + $2** USA ONLY SHIPPING/HANDLING

MasterCard VISA

BLECHMAN ENTERPRISES
7217 BERNADINE AVE., #10
CANOGA PARK, CA 91307

INFO/ORDER: 9AM - 5PM M.- SAT. PACIFIC TIME

818-346-7024

* Amway has not reviewed or endorsed these programs.

CHRISTMAS SPECIAL

12-16-20 MHZ.
HIGH SPEED
TURBO-286/386

- ▶ CPU-286-12/386-16 MHZ.
- ▶ 287/387 and 16-20 MHZ. opt.
- ▶ 1 MGB. on board. Max. 16 MGB.
- ▶ 2 Teac 1.2, or 360/3.5/720.
- ▶ 60 MGB. 28 MLS. with W.D. Contr.
- ▶ I/O, Serial, Parallel, Game.
- ▶ PGA, EGA, CGA, MGA, HERC. S. Sw.
- ▶ Deluxe Keybrd. 102/12 Funct. Keys.
- ▶ 200 W.P.S. Case. Digital Display.
- ▶ ☆☆ **TURBO-XT-W/Monitor...$899.00**

TURBO-286/386-$2,499.00/$3,499.00

COMPSTAR, INC.
(818) 842-4606
239 E. Alameda Ave., Burbank, CA 91502
TELEX: 797196 FAX: (818) 845-0739

CIRCLE 202 ON FREE INFORMATION CARD

IF YOU KNEW HOW EASY IT IS TO BUILD AN IBM® XT COMPATIBLE COMPUTER, YOU WOULDN'T PAY IBM® $1000 TO DO IT FOR YOU!

☐ Anyone can do it! No special skills required. ☐ **Takes less than 2 hours!** ☐ With a screw driver and my video tape you can save up to $1000 over the price of an assembled IBM®. ☐ Includes a list of suppliers of computer kits. (All kits have 90 day warranty and full compatibility) ☐ **MONEY BACK GUARANTEE** - If you aren't totally convinced that you can assemble your own computer after viewing my fail-safe, step by step video instructions, **you pay nothing!**

SEND $29 FOR VIDEO TAPE TO:
WOLFER PRODUCTIONS, Dept. R
10153½ Riverside Dr. #153
Toluca Lake, CA 91602
(California residents add 6.5% sales tax)

TURBO CHARGE YOUR PC WITH THE ACCELERATOR 286

Replaces your PC, XT, or compatibles 8088 with an 80286 processor. It includes 8K of zero-wait-state cache memory, and your software can run up to 7 times faster. So why buy an AT? Upgrade your PC with the Accelerator. **$275**

VIDEO SHARING DEVICE **$59.95**

Transmits color computer/VCR signal to any TV at home without cables. Simply plug it into com-puter's composite outlet or VCR's jacks. This is the least expensive way of upgrading your system to color without buying a color monitor. Minor assembly required.

PC Plus Corp. Monterey Park
California 91754
710 E. Garvey Ave.
(818)571-6991

CIRCLE 201 ON FREE INFORMATION CARD

processor's socket could lead to problems later, so we'll do it another way. The data bus is more accessible at the two EPROM sockets (IC20 and IC27) and at the two static RAM sockets (IC21 and IC28). The wiring diagram of the EPROM and static-RAM circuitry is shown in Fig. 3. Install a 28-pin socket at IC27 and a 24-pin socket at IC21.

The D0–D7 lines of the data bus are connected to IC27 via pins 11–13 and pins 15-19, and pin 14 is conveniently grounded. So we need to short pins 11–19 together. In the same way, the D8–D15 lines of the data bus are connected to IC21 via pins 9–11 and 13–17, and pin 12 is conveniently grounded. So we need to together pins 9–17 together.

On both IC21 and IC27, those are the bottom four pins on the left and the bottom five pins on the right. Instead of soldering any wires, take a strip of Molex Soldercon pins and insert them into the sockets as shown in Fig. 4, with four pins on the left, five pins on the right, and a section of six or so pins bent in a U shape below. Molex pins are normally sold as an inexpensive substitute for sockets; they consist of individual clips joined by a perforated carrier strip that is normally broken off after soldering the pins to the PC board. In our case, we insert the thin pins into the socket, and use the entire strip as one big short circuit.

Now plug in the IC's, being careful not to bend pins, and turn on the power. Connect your LED probe to pin 52 of the 68000, which is address line A23. If all is well, the LED will light for about 2 seconds, go off for about 2 seconds, and so on. (That flashing rate will be faster if your PT-68K's clock is faster than 16 MHz.)

If the LED does not flash at the expected rate, recheck the signal at every pin of the 68000. Look especially for a low on $\overline{\text{DTACK}}$, lows on all data lines, highs on $\overline{\text{BERR}}$, $\overline{\text{HALT}}$, and $\overline{\text{RESET}}$. When looking at the clock signal, the LED should be slightly dim.

If the LED flashes as expected, all is probably well. What's happening is that the 68000 is racing through memory (or what it thinks is memory, all 16 megabytes worth), executing *or* instructions at maximum speed, one instruction per microsecond. One complete run through four million instructions (4,194,304 instructions, to be exact) therefore will take slightly more than four seconds.

During that time, the address bus is counting off the addresses where the 68000 thinks those instructions are coming from. If you look at the address bus you see that A1 alternates 0, 1, 0, 1, 0, 1, ... A2 also alternates, but slower: 0, 0, 1, 1, 0, 0, ...; and so on.

Figure 5 shows the waveforms present on the upper eight bits of the address bus. When the LED probe is connected to A23, it flashes on and off once every four seconds. If it is connected to A22, it repeats once every two seconds; on A21 it repeats once per second. As we go down to A20, A19, and so on, it flashes faster and faster, until at A16 (pin 44) it flashes so fast (about 32 times per second) that we can barely see it flicker; A15, flashing about 64 times per second, looks absolutely steady.

Last, if you have access to an oscilloscope or frequency counter, examine each address line to make sure that its frequency is half that of its next higher neighbor. This ensures that there are no shorts between adjacent address lines. If you use a frequency counter, keep in mind that real-world waveforms are not as well defined as the idealized ones shown in Fig. 12, so some counters may have difficulty properly counting the frequency of such a square wave.)

Next time we'll add address decoders, ROM and RAM. See you then.

Ordering Information

Complete details were given in part one (in the October issue). To summarize: The basic kit (PT1, $200) contains all parts except power supply, case, and video terminal or personal computer to get a small system (ROM monitor, 2K RAM) up and running. The full basic system (PT68K, $460) includes 512K of dynamic RAM, floppy-disk controller, parallel port, battery-backed clock/calendar, and three PC-compatible expansion slots. To order, or for more information, contact Peripheral Technology, 1480 Terrell Mill Road #870, Marietta, GA 30067, (404) 984-0742.

FLOPPY-DISK DATA STORAGE

continued from page 94

illegitimate software floating around, so it seems that something has to be done about basic human nature before copy protection will cease to be an issue.

Even though there are obvious (and subtle) differences between the hardware and software comprising various types of computers, the basic approach to copy protection is the same: Make the disk unreadable by the standard DOS. It's easy to do because any DOS must make a number of assumptions about disk format before it tries to read or write information. It must assume, for example, that it's going to find tracks formatted in a particular way, that each one will contain a specific number of sectors, and that those sectors will contain data written in a predefined fashion. If any of those conditions aren't met, DOS will throw in the towel, and, instead of data, all you'll get is an error message. The point is that any disk that has data organized in a non-standard way must also have a non-standard way to read that data.

When Apple introduced its disk system in the late seventies, the company emphasized software rather than hardware. That was a departure from the norm, because most disk systems were and are built around a single-IC LSI controller. As a result, Apple disks were (and still are) unreadable by most other machines. However, Central Point Software's Option Board allows an IBM to read Apple disks, and many others as well.

Doing most of the disk control in software makes it simple to upgrade DOS. It also makes it easy for creative programmers to write copy-protection schemes that do strange things with the disk. That dependence on software, as we'll see, has produced methods of copy protection that are unique to the Apple.

Non-standard data formats

There are many methods of storing data in a non-standard format; we'll examine several in what follows. The most popular methods are these:

- Oddball track formatting
- Nibble counting
- Modified DOS
- Non-standard sectoring
- Unique data encryption
- Synchronized tracks
- Quarter tracks
- Spiral tracking

Of course, there are variations on those methods, and they're often used in combination. But attaining a good understanding of them will help you unravel any copy-protection scheme likely to come your way.

Those methods of copy protection are used on the Apple; due to differences in the IBM's disk-control hardware, it has fewer means of copy-protecting a disk. For example, the IBM cannot do quarter tracking. The most popular methods are:

- Oddball formatting
- Weak bits
- Laser burning

We'll examine those and other means of copy-protecting software next time.

COMPUTERDIGEST Volume 4

January 1987 — December 1987

Abbreviations: (C)Construction; (D)Department; (Ed)Editorial; (EW)Editor's Workbench; (HWR)Hardware Review; (LTR)Letter; (SWR)Software Review

A

Add a Disk Drive(Friedman)(C) **Apr 96**
All About Interfacing: Part 2 (Holtzman) **Jan 13**
Apparat Limbo II(EW) **Mar 94**

B

Biofeedback(L) **Feb 102**
Books(EW) **May 137, Nov 93**
Brooklyn Bridge(EW) **Sep 74**
Build
Clock Board for Your PC, A(C)(Martin) **Mar 97**
MC68000, The(C)(Schrader, Koenig, Voelzke) **Mar 101**
PT-68K, The(C)(Stark) **Oct 90**

C

CAD
Computer-Assisted Regulator Design (Cunkelman) **Feb 110**
Designing PC Boards on Your Computer (Grossblatt) **Jun 97, Aug 69**
CD-ROM **(EW) Mar 94, (EW) Mar 95, (EW) Apr 90**
Cauzin Softstrip System, The(Holtzman) **Apr 93**
Central Point Software: The Option Board(EW) **Oct 87**
Certificate Maker(SWR) **Feb 103**
Clock
Board for Your PC (C)(Martin) **Mar 97, (PCS) Mar 73**
Circuit for the PT-68K(C)(Stark) **Nov 101**
Clones
IBM-Compatible Clone Computer(C)(Flack) **Feb 104**
COMDEX Report(EW) **Sep 73**
Commodore Pulse Generator(C)(Barbarello) **Oct 96**
CompDes(EW) **May 136**
Computer Products(D) **Jan 4, Feb 102**
Computer-Assisted Regulator Design(Cunkelman) **Feb 110**
Computer-Controlled Robot(C) (Barbarello) **May 144, (PCS) May 67**
Concurrency(Stern) **Feb 112**
Construction
Add a Disk Drive(C) (Friedman) **Apr 96**
Clock Board for Your PC (C)(Martin) **Mar 97, (PCS) Mar 73**
Commodore Pulse Generator(C)(Barbarello) **Oct 96**
Computer-Controlled Robot(C) (Barbarello) **May 144, (PCS) May 67**
IBM Incompatible, Build This(C)(Holtzman) **Apr 98**
MC68000, Build The(C) (Schrader, Koenig, Voelzke) **Mar 101, May 138**
Micro-Floppy Retrofit(C)(Friedman) **Aug 67**
PT-68K, Build the(C)(Stark) **Oct 90, Nov 101, Dec 95**
Copy Protection
The Beginning of the End...(ED)(Wels) **Jan 3**

D

Data Storage, Floppy-Disk(Grossblatt) **Dec 91**
Debugging(EW) **Dec 87**
T-DebugPlus(EW) **Dec 87**
Turbo Pascal(EW) **Dec 87**
TurboSmith(EW) **Dec 87**
Designing PC Boards on Your Computer(Grossblatt) **Jun 97, Aug 69**
Direc-Link(EW) **Sep 74**
Do Me a Favor...(ED)(Wels) **Feb 101**

E

Editorial
Do Me a Favor...(ED)(Wels) **Feb 101**
The Beginning of the End...(ED)(Wels) **Jan 3**
Editor's Workbench **Mar 93, Apr 85, May 135, Jun 93, Jul 66, Aug 63, Sep 73, Oct 87, Nov 93, Dec 87**
Apparat Limbo II **Mar 94**
Books **May 137, Nov 93**
Brooklyn Bridge **Sep 74**
CD-ROM **Mar 94, Mar 95, Apr 90**
Central Point Software: The Option Board **Oct 87**
COMDEX Report **Sep 73**
CompDes **May 136**
Debugging **Dec 87**
Direc-Link **Sep 74**
Graphics **Sep 73**
Grolier's Electronic Encyclopedia **Apr 90**
Hardware Reviews **Mar 93**
Heathkit SK-203 Printer Buffer **Sep 76**
Hercules Graphics Card Plus **Apr 85**
IBM
Models 30 and 50 PC's **Aug 63**
Operating Systems: The M&M's **Jun 93**
Personal System/2 **Jul 66**
Keyboards **Jul 66**
Masters' Visible Computer: 8088 **Jun 95**
Memory Expansion **Mar 93**
Microsolutions' Matchpoint **Jun 94**
Mycroft Labs' Mite **Oct 88**
Orchid's Conquest **Mar 93**
PC-Sig's
Command-Line Editor **Mar 95**
PC-Outline **Mar 94**
PS/2 **Sep 73**
Portable Computers(Holtzman) **Nov 93**
Product Reviews **Mar 96, Apr 92, Jun 96, Oct 89, Nov 94**
68000 Update **Aug 63**
T-DebugPlus **Dec 87**
Tseng Lab's EVA/480 **May 135**
Turbo Pascal **Dec 87**
TurboSmith **Dec 87**
Vericomp's Breakthru/PC **Mar 94**

F

5-Volt Only: The Max 232(Kreuter) **Jan 10**
From Keypress to Scan Code(Holtzman) **Jul 70**

G

Graphics(EW) **Sep 73**
Graphics Co-Processors(Bernard) **Sep 82**
Grolier's Electronic Encyclopedia(EW) **Apr 90**

H

Hardware
Add a Disk Drive(C)(Friedman) **Apr 96**
All About Interfacing: Part 2(Holtzman) **Jan 13**
Build the MC68000(C) (Schrader, Koenig, Voelzke) **Mar 101, May 138**
Build the PT-68K(C)(Stark) **Oct 90, Nov 101, Dec 95**
Cauzin Softstrip System, The(Holtzman) **Apr 93**
Commodore Pulse Generator(C)(Barbarello) **Oct 96**
Concurrency(Stern) Feb 112
Editor's Workbench(EW) **Mar 93**
Graphics Co-Processors(Bernard) **Sep 82**
IBM-Compatible Clone Computer(C)(Flack) **Feb 104**
IBM Incompatible, Build This(C)(Holtzman) **Apr 98**
Floppy-Disk Data Storage(Grossblatt) **Dec 91**
MC68000, Build the(C) (Schrader, Koenig, Voelzke) **Mar 101, May 138**
Micro-Floppy Retrofit(C)(Friedman) **Aug 67**
Portable Computers(Holtzman) **Sep 77**
Heathkit SK-203 Printer Buffer(EW) **Sep 76**
Hercules Graphics Card Plus(EW) **Apr 85**

I

IBM
-Compatible Clone Computer(C)(Flack) **Feb 104**
-Incompatible, Build This(C)(Holtzman) **Apr 98**
Models 30 and 50 PC's(EW) **Aug 63**
Operating Systems: The M&M's(EW) **Jun 93**
Personal System/2(EW) **Jul 66**
Interfacing, All About; Part 2(Holtzman) **Jan 13**

J

J&M's Memory Minder(EW) **Aug 63**

K

Keyboards
Editor's Workbench **Jul 66**
From Keypress to Scan Code(Holtzman) **Jul 70**
Working with Surplus Keyboards(Grossblatt) **Jul 74**
Keprom, The(Holtzman) **Jun 100**

L

Letters(D) **Jan 4, Feb 102**
Look Inside the 6502, A(Solomon) **May 141**

M

Max 232: 5-Volt Only(Kreuter) **Jan 10**
MC68000, Build the(C) (Schrader, Koenig, Voelzke) **Mar 101, May 138**
Memory Expansion(EW) **Mar 93**
Micro-Floppy Retrofit(C)(Friedman) **Aug 67**
Microsolutions' Matchpoint(EW) **Jun 94**
Multitasking
Concurrency(Stern) **Feb 112**
Mycroft Labs' Mite(EW) **Oct 88**

O

Orchid's Conquest(EW) **Mar 93**

P

PC Boards, Designing on Your Computer(Grossblatt) **Jun 97**
PC-Sig's
Command-Line Editor(EW) **Mar 95**
PC-Outline(EW) **Mar 94**
Periscope Software(EW) **Dec 87**
Portable Computers(Holtzman) **Sep 77, (EW)Nov 93**
Product Reviews(EW) **Mar 96, Apr 92, Jun 96, Oct 89, Nov 94**
Programs
Commodore Pulse Generator(C)(Barbarello) **Oct 96**
Computer-Assisted Regulator Design (Cunkelman) **Feb 110**
TV Channel Frequency Program(Kiley) **Jan 12**
TVRO Antenna Pointer(Tyson)
PS/2(EW) **Sep 73**
PT-68K, Build the(C)(Stark) **Oct 90, Nov 101, Dec 95**

R

Robot, Computer-Controlled(C) (Barbarello) **May 144, (PCS) May 67**

S

Software
Editor's Workbench **Mar 94, Apr 90, May 136, Jun 93, Aug 63, Sep 74, Oct 88, Nov 93, Dec 87**
Designing PC Boards on Your Computer: Part 2(Grossblatt) **Aug 69**
Masters' Visible Computer: 8088(EW) **Jun 95**
Periscope(EW) **Dec 87**
Pirates, Sinking the(Holtzman) **Jun 100**
Review(D) **Jan 5, Feb 103**
Certificate Maker(SWR) **Feb 103**
Webster's On-Line Thesaurus(SWR) **Jan 5**
6800
MC68000, Build the(C) (Schrader, Koenig, Voelzke) **Mar 101, May 138**
PT-68K, Build the(C)(Stark) **Oct 90, Nov 101, Dec 95**
68K Update(EW) **Aug 63**
6502, A Look Inside the(Solomon) **May 141**

T

Test Equipment
Commodore Pulse Generator(C)(Barbarello) **Oct 96**
Tseng Lab's EVA/480(EW) **May 135**
TV Channel Frequency Program(Kiley) **Jan 12**
TVRO Antenna Pointer Program(C)(Tyson) **Jan 6**

V

Vericomp's Breakthru/PC(EW) **Mar 94**

W

Webster's On-Line Thesaurus(SWR) **Jan 5**
Working with Surplus Keyboards(Grossblatt) **Jul 74**

S

Satellite TV(D)(Cooper) Jan 4, Feb 83, Mar 78, May 77, Jul 62, Aug 28, Sep 87, Oct 80, Dec 74
- High-Definition DBS Jul 62
- International Connection, The Sep 87, Oct 80
- Is HDTV the Key to an International Standard? Aug 28
- Practical Descrambling Feb 83
- Videocipher Has Been Cracked Jan 4
- What's Next Dec 74
- Why Videocipher is Dead Mar 78
- Zits Fraud, The Jun 76
- Zombies, Zits, and Zoweee! May 77

SCA
- Decoder(ARE) Jun 10
- FM-Stereo Receiver(C)(Sheets and Graf) Aug 39, Sep 46, (PCS) Sep 69 (LTR) Oct 15

SCR's
- Using Triac's and SCR's(Marston) Sep 64
- Working with Triacs and SCR's(Marston) Oct 64

Scanner
- Booster for(ARE) Mar 12
- Regency Informant Scanning Receiver(C) Aug 16

Seismic Discussion Net(LTR) May 13

Semiconductors, Testing(Byers) Feb 58, Mar 71, Apr 62 May 59, Jun 61, Aug 12, Sep 61, Nov 115 (LTR) Aug 12, (LTR) Nov 8

Sencore LC75 Z Meter II(ER) Apr 24

Sequential Flasher(NI)(Ciric) Feb 36

Service
- **Clinic(D)(Darr)** Jan 88, May 125, Sep 30
  - Funny Pictures May 125
  - Leakage and Psychology Jan 88
  - Quirks and Queerities Sep 30
- **Questions(D)(Darr)** Jan 90
- Log: Surface-Mount Components(Poe) Nov 32

Shortwave Converter
- New Life for Old Car Radios(C)(McClellan) May 42, Jun 50, (PCS) Jun 71, (LTR) Jul 12, (LTR) Oct 15

Simple Multi-Tone Generator(NI)(Khan) Nov 31

Smart House
- Home of the Future, The(MacFadyen) May 115

SMT Projects(See SURFACE MOUNT TECHNOLOGY)

Soldering: Old Techniques and New Technology(Martin) May 47, (LTR) Aug 12, (LTR) Sep 13

Solid-State (See also STATE OF SOLID STATE)
- Technology in the 21st Century(Gregory) May 97
- Tube Substitutes(ARE) Apr 8

Sound-Effects Generator(NI)(Tupue) Jun 40

Soundproofing
- Unwanted Sounds(AUD)(Klein) Jun 78

Speakers, Magnetically Shielded(AUD)(Klein) Oct 83

Spectrum Analyzer, Portable(See TEST EQUIPMENT)

Speedometer, Digital(C)(Ortman) Jul 47, (PCS) Jul 79, (LTR) Nov 8

Spread Spectrum Communications, All About(McDermott) Apr 55

State of Solid State(D)(Scott) Feb 92, Mar 84, Apr 80 May 129, Jun 87, Sep 94, Nov 124, Dec 42
- Bang-Bang IC, A Nov 124
- Electronic Potentiometer, An Dec 42
- Instrumentation Amplifiers Jun 87
- Long-Time Timer May 129
- Micropower Op-Amp Sep 94
- Temperature Transducer Mar 84
- Tone Generator IC's Apr 80
- Transformerless 5-Volt Regulator Feb 92

Stereo(See also AUDIO)
- Imagery amd Psychoacoustics(AUD)(Klein) Mar 80
- Spatial Imaging(AUD)(Klein) Nov 33
- Spread Circuit(ARE)(Scott) Feb 12
- TV Decoder(C)(Templin) Jan 37, Feb 51
- Why Stereo Doesn't Work(AUD)(Klein) Apr 68

Strain-Gage Transducers(Wood) Dec 61

Sunrise/Sunset Simulator(ARE)(Scott) Aug 8

Surface-Mount Technology
- Surface-Mounted Components(LTR) Feb 16
- Conductive Inks and Adhesives(Mims) Nov 81
- Hand-Soldering SMC's(Mims) Nov 71
- Industrial SMT Assembly(Mims) Nov 65
- Introduction to SMT(Mims) Nov 59
- **SMT Project**
  - Business-Card Tone Generator(C)(Mims) Nov 85
  - I-R Remote on a Keychain(C)(Mims) Nov 77
  - LED Flasher(C)(Mims) Nov 73
  - Light Meter(C)(Mims) Nov 75
- Resource Directory Nov 89
- Service Log(Poe) Nov 32

Surround Sound, Resurgence of(AUD)(Klein) May 74

T

Telegraph, and WWI(AR)(Fitch) Feb 88

Telephone
- Blue Box and Ma Bell, The(Friedman) Nov 49
- Inside Cellular Telephone(Bernard) Sep 53
- Nine-Station Intercom(C)(Morrison) Jan 57, (PCS) Jan 67
- Phonlink Interactive Remote Control(C)(Roseth) May 39, Jun 53, (PCS) Jun 71

Television(See VIDEO)

Tesla, Father of Radio (LTR) Jun 14

Test Equipment
- Avcom PSA-35A Portable Spectrum Analyzer(ER) Jul 15
- Beckman Industrial DM 800 DMM(ER) Jan 71
- Finding Cable Faults(Martin) Mar 66
- Fluke LCA-10 Line Current Test Adapter(ER) Mar 28
- How to Analyze Waveforms(Carey) Dec 59
- In-Circuit Digital IC Tester(C)(Green) Nov 43
- Leader LCD-100 Portable DMM/ Storage Oscilloscope(ER) Jun 21
- NCM's Video Wonderbox(ER) Nov 14
- Oscilloscopes, Using the New Generation(Diller) Feb 55
- Philips Compact Disc Test Set(ER) Feb 27
- Poor Man's Storage Scope(Bernard) Nov 113
- Sencore LC75 "Z Meter II"(ER) Apr 24

Temperature Transducer(SOSS)(Scott) Mar 84

Testing Semiconductors(Byers) Feb 58, Mar 71, Apr 62 May 59, Jun 61, Aug 12, Sep 61, Nov 115 (LTR) Aug 12, (LTR) Nov 8

TI/P-CAD PAL Starter Kit(ER) Mar 32

Timer, Versatile Digital(C)(Ortman) Aug 45, (PCS) Aug 75

Timing Light Modification(ARE) May 10

Tone Generator IC's(SOSS)(Scott) Apr 80

Tool Organizer, Maszota(LTR) Mar 16

Transducers, Strain-Gage(Wood) Dec 61

Transformerless 5-Volt Regulator(SOSS)(Scott) Feb 92

Transistor Amplifier Design(Cunkelman) Aug 55

Triacs
- Using Triac's and SCR's(Marston) Sep 64
- Working With Triacs and SCR's(Marston) Oct 64

Tri-Mode Cable-TV Scrambling(C)(Coffell) Feb 43, (PCS) Feb 78 (LTR) May 13

Trigger Pulses(DN)(Grossblatt) May 121

TSM 201 Clock Module Nov 122

Tunable IF(COMC)(Friedman) Jan 86

TV Signal Descrambling(C)(Sheets and Graf) Jan 53, Mar 63, Jul 58

2001
- Automotive World of the 21st Century, The(Petersen) May 91
- Communications in 2001: The Third Age of Video(Judice) May 102
- Dear Sue,...(Oliver) May 84
- Energy Technology in the 21st Century(Kuznetsov) May 107
- Home of the Future, The(MacFadyen) May 115
- Looking Into the Future(Clarke) May 81
- Medical Technology in the 21st Century(Fish) May 112
- Robot in the 21st Century, The(Asimov) May 99
- Solid-State Technology in The 21st Century(Gregory) May 97
- Future of Artificial Intelligence, The(Heilmeier) May 85
- Welcome to the Twenty-First Century(ED)(Fenton) May 4

U

Under-Voltage Monitor, An(DN)(Grossblatt) Nov 41

Understanding Data Sheets of RF Power Transistors(Dye) Nov 104

Universal Wireless Remote Control/Stereo TV Tuner(ER) May 24

Using
- New Generation Oscilloscopes, The(Diller) Feb 55
- Polapulse Battery, The(Blechman) Feb 61, (LTR) Jun 14
- RE-BBS, The May 122
- Triac's and SCR's(Marston) Sep 64

V

Vacuum Tubes(LTR) Feb 16

Versatile Digital Timer(C)(Ortman) Aug 45, (PCS) Aug 75

VHSIC, The Evolution of(Grossblatt) Mar 59

Video
- Communications in 2001-The Third Age of Video(Judice) May 102
- Black Vertical Bars(Shane) Jun 70
- Buyer's Guide to Camcorders(Vizard) Mar 47
- Effects Generator(C)(Sheets and Graf) Sep 41, Oct 48, (PCS) Oct 75
- Palette(C)(Sheets and Graf) Sep 41, Oct 43 (PCS) Oct 75
- High Definition TV(Bernard) Aug 48
- **News(D)(Lachenbruch)** Jan 16, Feb 6, Mar 6, Apr 7, May 8, Jun 8, Jul 6, Aug 6, Sep 6, Oct 6, Nov 6, Dec 12
- Stereo TV Decoder(C)(Templin) Jan 37, Feb 51, (PCS) Feb 79
- Tri-Mode Cable-TV Scrambling(C)(Coffell) Feb 43, (PCS) Feb 79, (LTR) May 13
- TV Signal Descrambling(C)(Sheets and Graf) Jan 53, Mar 63, (PCS) Mar 73, Jul 58
- Universal Wireless Remote Control/Stereo TV Tuner(ER) May 24
- VHSIC, The Evolution of(Grossblatt) Mar 59
- **Videocipher**
  - Practical Descrambling(STV)(Cooper) Feb 83
  - The Zits Fraud(STV)(Cooper) Jun 76
  - Videocipher Has Been Cracked(STV)(Cooper) Jan 4
  - Why Videocipher is Dead(STV)(Cooper) Mar 78
  - Zombies, Zits, and Zowee!(STV)(Cooper) May 77

Voltage Transformers(LTR) Aug 12

W

Waveforms, How to Analyze(Carey) Dec 59

Welcome to the Twenty-First Century(ED)(Fenton) May 4

What's
- A Gate-Turnoff Rectifier?(ARE) May 10
- New in Solid State(Scott) Jan 61
- **News(D)** Jan 14, Feb 4, Mar 4, Apr 4, May 6, Jun 6, Jul 5, Aug 4, Sep 4, Nov 4, Dec 6

Working With Triacs and SCR's(Marston) Oct 64